Lehigh University
Lehigh Preserve

Theses and Dissertations

1987

An X.25 level-2 communications board /

Michael R. Kost Jr.
Lehigh University

Follow this and additional works at: https://preserve.lehigh.edu/etd

 Part of the Electrical and Computer Engineering Commons

Recommended Citation
Kost, Michael R. Jr., "An X.25 level-2 communications board /" (1987). *Theses and Dissertations*. 4765.
https://preserve.lehigh.edu/etd/4765

This Thesis is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.

AN X.25 LEVEL-2 COMMUNICATIONS BOARD

by

MICHAEL R. KOST JR.

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computer Science

Lehigh University

May 1987

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 13, 1987
(date)

Donald J. Hillman
Professor in Charge

Donald J. Hillman
Head, Division of Computer Science

L. J. Varveris
Chairman, Department of CSEE

Contents

ABSTRACT

This thesis describes a communication board which performs X.25 level-2 Protocol Control. The level-2 protocol is done in hardware by the AT&T T7100 & T7102 Integrated Circuits.

This thesis describes what X.25 is, how it relates to the world of data communications and also how it is implemented in our system. Parts of this communication system are described as well as how to initialize the protocol controllers in software.

CHAPTER ONE

INTRODUCTION

Data Communication is becoming an integral part of the computing environment. The more computers entrench themselves in our society the more important the transfer of information from one point to another. With hundreds of different types of computers and operating systems, the problem becomes more complex and harder to transfer information. A remedy to this problem is to standardize the information transfer problem. So that no matter what form the information takes it will be able to transfer from one point to another unaltered.One current standard in data communication is X.25 and recently part of this standard was done in hardware by AT&T in the form of an integrated circuit.

In the following chapters we will explain more about data communication specifically X.25. This paper also describes our first attempt to design a system using this hardware implementation and the system software required to run it. The system which we developed is a communication protocol board which implements X.25 level-2. The level two protocol implemented in hardware will be provided by AT&T's T7100(XPC-8) and T7102 protocol controllers (collectively, the XPC's).

In this system the hardware and the software were designed so that they could work together. The design process

involved hardware/software trade-off issues which will be explained as required.

The system is referred to as the COMMUNICATION BOARD throughout this paper.

The overview of the paper will explain X.25 as defined by the CCITT committee. Specifically what X.25 level-2 LAPB protocol is and how it is implemented in hardware IC and the software needed to run this chip.

The remainder of the paper is devoted to the communication board which is a large hardware/software project based on our first attempt to use the XPC's devices. The hardware aspects of the project involved logic design, and hardware debugging. The software aspects involve the implementation of the "C" compiler on the host unix system, software adjustments of the CPU monitor and the software development of the code to drive the devices on the communication board. I will attempt to explain my role in its development, and the areas where I contributed to its creation.

CHAPTER TWO

OVERVIEW

2.0) PACKET SWITCHING

Communication networks have long used switching to reduce the number of connections linking all users and to allow data processing equipment to efficiently share the total network resources. Voice transmission, such as during a telephone call, requires a dedicated path between users for the duration of a call. Information is transferred in real time without intentional change; the result is a transparent network to the users. In contrast, data communication involves a transactional approach, whereby the information is introduced into the network as a complete entity.

Whereas voice calls are of relatively long duration and require small system delay to avoid speech distortion, data messages are generally of short duration and can tolerate delay. Overall system bandwidth or capacity can be utilized more efficiently when it can assigned as needed. The greater flexibility of choosing bit rates for transactional networks leads to more efficient use of network capacity for data communication than for voice communication. The disadvantages of the transactional approach are that the network requires data processing and a storage facility, and that complex routing and control

procedures (i.e., protocol) must be employed.

In packet switching, information is exchanged in the form of blocks of limited size, or "packets." Long messages are divided into several packets that are transmitted through a series of transactions and then reassembled at the destination to reconstitute the original message. Such network resources as memory, processing power, bandwidth, etc., are shared among many users. The price for this efficiency is network complexity, since data buffers are needed at each network node. Signaling, control functions, and information transfer also require elaborate procedures. To reduce design complexities and facilitate data communications, packet switched networks are usually organized as a series of layers built one upon the other (see Figure 1). Each layer offers certain services to the one just on top of it, while shielding specific details concerning implementation of these services. In the open systems interconnection (OSI) proposed by the International Standards Organization (ISO), there are seven such layers:

1. *The Physical Layer* (or Level-1) is concerned with mechanical, electrical, and timing procedures for the transmission of data bits over a physical medium. This layer includes such hardware as modems and communications lines.

2. *The Link Layer* (or Level-2) assures that frames are transported reliably across a data link. This is done by providing frame numbering and cyclic redundancy check, together with link layer procedures that ensure data integrity, such as flow control, error detection, and retransmission.

3. *The Network Layer* (or Level-3) is concerned with end-to-end transmission, hence establishing end-to-end connections, and routes higher-level information through the data communications network. It provided transport entities with independence from routing and switching considerations.

4. *The Transport Layer* (or Level-4) exists to provide a universal transport service in association with the underlying services provided by lower layers. It provides transparent transfer of data between session entities and relieves these session entities from any concern with the detailed way in which reliable and cost-effective transfer of data is achieved.

5. *The Session Layer* (or Level-5) assists in the support of the interactions between cooperating presentation entities. To do this, the session layer is responsible for binding two presentation entities into a relationship and

unbinding them (session administration service); it is also responsible for the control of data exchange, delimiting, and synchronizing data operations between two representation entities (session dialogue service).

6. *The Presentation Layer* (or Level-6) provides a set of services that may be selected by the applications layer to enable it to interpret the meaning of the data exchanged. This may include data conversion, encryption, or decryption.

7. *The Applications Layer* (or Level-7) is the highest layer of the OSI structure. Protocols of this layer directly serve the end user by providing the distributed information service appropriate to an application, to its management, and to its systems management.

2.1) WHAT IS THE X.25?

To ensure data equipment compatibility, CCITT (Consulting Committee for International Telephone and Telegraph) Recommendation X.25 has defined a standard interface protocol between packet network data communication equipment (DCEs) and packet mode data terminal equipment (DTEs). (The term DXE refers to either a DTE or a DCE.) This recommendation, first adopted in 1976, was significantly revised both in 1980 and 1984. In order to clarify DTE operations on an X.25 network, ISO has also

specified X.25 LAPB-compatible DTE procedures in ISO Documents 7776 and 8208.

CCITT Recommendation X.25 and ISO 7776 define three levels of communications: the physical, link, and packet levels. While the physical and link levels of X.25 corresponds directly to layers 1 and 2 of the OSI reference model, the packet level of X.25 includes aspects of both the network and transport layers (layers 3 and 4) of the OSI reference model.

2.1.1) X.25 LEVEL-1 THE PHYSICAL LEVEL

The X.25 physical level deals with the representation of data bits, timing aspects, and physical contact by referring ro recommendations X.21 and X.21 bis. The RS-232C EIA standard conforms to X.21 bis and the V-series interfaces.)

2.1.2) X.25 LEVEL-2 THE LINK LEVEL

The link level elements used in X.25 are the link access procedures (LAP and LAPB). These procedures are compatible with a class of standard bit-oriented protocols, namely, HDLC (high-level data link control), ADCCP (advanced data communication control procedure), and SDLC (synchronous data link control). Only the LAPB procedure, however, is required in all public switched data networks.

LAPB defines procedures for link set-up, information transfer, and link disconnect. Through a system of acknowledgments, error detection, and retransmission, LAPB

provides an error-free data link despite the unreliability of the physical medium. At a receiver, information is delivered in units (or packets) to the packet layer without loss or duplication and in the same sequence of transmission.

Frames are delimited by specific bit patterns (flags) the also provide a means for link level synchronization. The LAPB frame structure consists of a link-level header, and information field, and a check code. The header indicated whether the frame is a command or a response and distinguishes between frame types (i.e., information frames, supervisory frames, or unnumbered frames). The information field is present only in information frames and is used to transfer packets across the data link. To detect physical level transmission errors, a 16-bit cyclic redundancy code is appended to each frame.

Certain frames also contain a send and/or receive sequence number. The send sequence number is used only in information frames and identifies the transmitted frame, in modulo-8 arithmetic, so that out-to-sequence frames can be detected. The receive sequence number is a part of both information and supervisory frames, and is used to acknowledge the receipt of information frames sent by the remote DXE.

When level-3 packets are ready for transmission, they are sent to level-2 to be queued for transmission over the data

link. A copy of each packet is maintained in the transmitting station until its acknowledgment is received. When the remote DXE acknowledges receipt of one or more information frames, the packets associated with those information frames may be removed from storage.

Whenever an information frame is sent, a timer is started. If an acknowledgement for that frame is not received before the timer expires, the local DXE queries the remote DXE for that acknowledgement by sending an appropriate supervisory frame. Queries of the remote DXE for the acknowledgement continue periodically for a preprogrammed number of times before alternative actions are taken.

When DXE receives packets faster than the higher level can provide buffer space to store them, a busy condition is indicated by the local DXE by transmitting a receiver not ready (RNR) frame. A DXE, notified of a remote DXE busy condition, temporarily discontinues the transmission of information frames until the busy condition clears.

When an out-of-sequence frame (i.e., an information frame with a send sequence number error) is received, a reject (REJ) frame is sent to the remote DXE to inform it of the event and to instruct it to begin retransmission of information frames, starting with the earliest unacknowledged frame. A timing function is also used for this condition to protect against the loss of frames.

A frame reject (FRMR) frame is transmitted whenever an invalid frame is received that cannot be corrected by retransmission. It is produced when one of more of the following conditions occurs in a received frame:

- an invalid receive sequence number

-an information frame with too long an information field

- an invalid control field

- a supervisory or unnumbered frame with a information field.

The frame reject condition is regarded as a serious link error condition; therefore, the link layer is reset before information transfer can continue.

2.1.3) X.25 LEVEL-3 THE PACKET LEVEL

The X.25 packet level defines the procedures and formats of the service, such as call establishment and data transfer, using the concept of logical channels for multiplexing calls to utilize the link bandwidth more efficiently. The 1984 version of X.25 allows up to 4095 logical channels on an individual interface (channel number zero is reserved for control packets that affect the entire interface). Therefore, many individual low-speed terminals can be multiplexed on a single higher-speed digital channel at each DTE/DCE interface, using the logical channel number for identification. Theoretically, with modulo-8 transmission, there could be up to 7 outstanding data

packets, each carrying up to 4096 octets on each of the 4095 logical channels. However, the number of logical channels supported is limited by available buffer storage, processing power, etc.

The allocation of logical channels can be accomplished in either a static or a dynamic (i.e., for each call) fashion. The latter case is a virtual circuit that emulates a point-to-point private line. The latter case is a virtual call, which requires call establishment and call clearing procedures. In either case, the network delivers packets to a destination in the order of transmission by the originating DTE. The specification of which channels are reserved for virtual calls and which are for permanent virtual circuits is determined at subscription time. In the 1984 version of X.25, these channels ranges may also be dynamically changed by using registration procedures. Furthermore, packet level procedures may be tailored to individual needs by using optional facilities.

X.25 allows customers to negotiate the selection of special service features or facilities at call establishment. All networks must provide essential facilities but may also offer additional ones. In either case, the user selects which facilities, if any, to use.

The operation of the packet level in X.25 is detailed below. After the link has been set up, the calling DTE must first establish a virtual circuit (in the case of

virtual calls) between itself and the destination. The calling DTE chooses one idle logical channel (except zero or those reserved for incoming calls only) from those that have been reserved for virtual calls. To minimize the risk of call collision, the logical channel used should be the one with the highest number among all available channels that have been assigned for two-way or one-way outgoing virtual calls. If the called DTE informs the network that it accepts the call, the virtual circuit is established and both DTEs may now use the full duplex virtual circuit to exchange data packets. The packet-level window size controls the maximum number of packets that can be waiting for acknowledgement on a single channel. In contrast, the level-2 window size affects all logical channels of the interface.

A call is normally terminated when either DTE issues a clear request packet. Note that, since permanent virtual circuits are always established, call establishment and clearing procedures are not used on the corresponding logical channels.

To illustrate the above by specific exchanges of X.25 frames/packets, we shall assume a passive DCE and an active DTE. This means that the DCE will either initiate the link nor the packet levels and that the DTE will initiate both. A typical operation can be described in a simplified fashion as follows:

1. To initialize the link, the DTE sends an SABM (set asynchronous balanced mode) frame, To accept the invitation, the DCE should return a UA (unnumbered acknowledgment) frame.

2. The DTE next initializes the packet level. It sends a restart request packet to initialize all logical channels on the link. The DCE responds with a restart confirmation packet.

3. At call establishment, the DTE builds a call request packet and passes it to the DCE.

4. The network routes the packet to the destination DCE according to the procedures defined by the network internal protocol (not specified by X.25). The destination DCE then transmits an incoming call packet to the destination DTE and uses the channel with the lowest available logical number among those assigned for two-way or one-way incoming virtual calls.

5. If the destination DTE wishes to accept the call, it responds by using a call accepted packet to its DCE. The network then delivers a call connected packet to the originating DTE.

6. If the destination DTE does not accept the call, it returns a clear request packet. (The packet may also contain a code specifying why the call has been refused.) The network transmits to the originating DTE a clear

indication packet to which the DTE responds by returning a clear confirmation packet.

7. If the connection is established, data packets are exchanged. Packet level send and receive sequence numbers are used in a fashion similar to that found in the link level to assure end-to-end data integrity. If a DTE receives packets faster than the higher protocol levels can process them, a busy condition may be indicated.

8. To prevent deadlocks several timers are implemented in the packet level. Further information is available in the appropriate X.25 document.

Although X.25 is concerned with synchronous communications, asynchronous terminals can interface to X.25 networks through a packet assembler/disassembler (PAD). Other CCITT recommendations specify this asynchronous interface. Recommendation X.3 specifies a set of parameters used by the PAD to control asynchronous terminals. Users of asynchronous terminals, in turn, may modify the individual parameters according to procedures specified by Recommendation X.28. The remote X.25 DTE may also modify the parameters according to the procedure identified by Recommendation X.29. X.25 packets that control the PAD are data qualified packets, i.e., they have their Q-bit (bit number 8 in the first octet of a data packet header) set (1) by the transport and higher layers. In contrast, the Q-bit is always 0 for data packets containing user data.

CHAPTER THREE

X.25 LAPB

This section will explain the the implementations of LAPB protocol by the XPC's. Included is the XPC's role in LAPB communication.

3.0) ROLE OF THE XPC'S IN X.25 LAPB COMMUNICATION

The XPC'S offers a multitude of programmable features. Among these features are programmable command and response address fields, a programmable window size (k), a programmable maximum information field length (N1), four independently programmable link assurance timers (T1 - T4), and a programmable retransmission counter (N2). A number of auxiliary features, such as identification exchange, loopback test modes, and system bus configuration (Intel or Motorola formats), are also programmable.

The XPC'S automatically maintains the link by handling supervisory and unnumbered frames without user intervention, interrupting the host CPU only when it requires attention. Typical interrupts are "transmitted block acknowledged" and "packet received". "Transmitted block acknowledged" informs the host CPU that transmitted data has been acknowledged by the remote DXE, enabling the host CPU to free the associated transmit data buffers. "Packet received" informs the host that data has been received and written to the appropriate receive data

buffer. Other interrupts inform the host of link conditions and problems with the physical link.

The interface between the XPC'S and the physical layer (level 1) consists of seven signals: request-to-send, clear-to-send, transmit data, receive data, transmit clock, receive clock, and carrier detect. These signals assume the usual RS-232 definitions.

DMA is used to support the XPC'S to host interface. The host defines regions of system memory for use as transmit data buffers, receive data buffers, and buffer management tables. The XPC'S uses the buffer management tables to indirectly access the transmit and receive data buffers, and to control the reallocation of these buffers. The host is able to monitor the management tables to aid in the transfer of data to the from the data buffers.

3.1) XID CAPABILITIES FOR X.32

Neither CCITT X.25 nor ISO 7776 LAPB use the XID command and response frames. However, these are required for CCITT X.32 (which is a variant of X.25 operating over dialup or circuit switched data network access links) and BX.25. The XID frames allow the two ends of the link to verify each others identities before a link setup. The AT&T T7100 and T7102 automatically handle an XID exchange.

3.3) ACTIVE VERSUS PASSIVE LINK SETUP

X.25 LAPB is a balanced point-to-point data link protocol, with both ends of the link potentially having the

indentical capability. This means that during link setup both the DTE and DCE may attempt to set up the link by sending as SABM command frame. Depending on which side gets an SABM out first, the link will be set up either by the DTE of DCE. If the DTE and DCE both send SABM at the same time, the SABM frames are said to collide and both the DTE and DCE are obliged to respond UA to complete link setup.

3.3) TRANSMISSION OF DM RESPONSE

When a receiving station is in the disconnected state, it should send the DM response for any valid command frame that carries P = 1, other than SABM. It should also respond DM to SABM if the station is not ready to set up the link. Some implementations may choose to ignore the incoming command frame rather than responding DM, but this is in violation of X.25 LAPB.

The consequences of using one method over the other is relatively minor. Returning DM allows the other station to know that the disconnected station did receive the command and was not ready to process the command. Otherwise, this situation may not be distinguishable from physical outages in the path.

The AT&T T7100 and T7102 will either send DM or send nothing at all, depending on control parameters set by the microprocessor, and whether or not the poll bit was set.

If the poll bit is set the AT&T devices will always respond.

3.4) TRANSMISSION OF DISC COMMANDS

The DISC Command frame is used to place the receiving station in the disconnected state. The AT&T T7100 and T7102 generate this command as a result of having their mandatory disconnect control bits set by software control. The generation of DISC affects the way the microprocessor drives the controller. For example, both idle link detection and retry limit exceeded are required for link initialization. Without software instruction the AT&T T7100 and T7102 automatically bring the link back up with SABM/UA, if set in the active link setup mode. (In passive mode, these devices wait for SABM from the other station).

3.5) NON-OCTET ALIGNMENT

The CCITT X.25 Recommendation and the ISO 7776 and 8208 International Standards state that for many Packet Switched Networks the Information-fields must contain an integral number of octets aligned on an octet boundary. Address, control, and FCS fields are always octet aligned. The AT&T T7100 does not offer the option of non-octet alignment. This feature is important only in non-X.25 systems where the basic unit of information transferred is not an octet. It is highly unlikely that CCITT X.25 will be changed in the future to allow this.

3.6) WINDOW OPERATION

For those X.25 LAPB links that use the non-extended control field format, a modulo-8 frame sequence numbering scheme is used. This automatically provides a maximum limit of seven unacknowledged frames outstanding in each direction of the link. This results in a window size of seven. Indeed, most non-extended control field links use a window size seven. With extended control field links, the modulo-128 frame sequence numbering scheme results in a maximum window size of 127. Although the reason for implementing an extended control field format link is the requirement for a window size greater than seven, having 127 possible frames outstanding may be impractical. (For example, all the frame buffers would have to be saved until the acknowledgements actually came in). Thus, being able to specify a window size smaller than the maximum accorded by the control field format is desirable.

The AT&T T7100 provides a variable window size even though it supports only modulo-8 operations. The AT&T T7102 allow the specification of window sizes smaller than the maximum in both extended and nonextended control field formats.

3.7) MODULO-128 FRAME NUMBERING

For systems where transmission propagation delays may be high, the use of a modulo-8 frame numbering scheme may result in nonoptimum use of the data channel. It is possible, on such data channels, for the transmitter to

have sent seven frames and be forced to stop because all frames are still in transit (and therefore unacknowledged). The use of the modulo-128 frame numbering allows up to 127 frames to remain outstanding before the transmitter must stop.

3.8) 32-BIT FCS

This feature strengthens the data link's ability to detect errors. The 32-bit FCS is not part of CCITT X.25 LAPB but is an option in ISO counterpart International Standard 7776.

3.9) TIMER CAPABILITIES

Timer capabilities range from the provision of the basic T1 waiting acknowledgement timer to the provision of all four timers defined in ISO 776 (AT&T 7100). The AT&T T7102 provides the T1, T3 idle link timer and a T4 inactive link timer.

The T1 timer is essential, however, if T4 is not provided, it can be implemented by microprocessor software when the idle link condition is detected and reported.

T4 (maximum time without exchanging frames) may be useful depending on what action is taken on T4 run-out. Ideally, T4 run-out should cause the controller to transmit a supervisory frame with P = 1, as a deep-alive function. If the other station is malfunctioning, but still able to flag fill, the action of T1xN2 will then invoke error reporting/recovery actions.

T2 (the maximum wait time before transmitting an acknowledgement) is not useful in a VLSI X.25 link level implementation. Those devices that do not provide T2 (T7102) behave as though T2 = 0. That is, all acknowledgments are transmitted as soon as possible. In software implementations of the X.25 link layer, it may be advantageous to wait a period of time before acknowledging. This allows the microprocessor to do other processing during the period of T2; including the construction of an outbound I-frame onto which the acknowledgment can be piggy-backed.

3.10) CONTROL/STATUS HANDLING

The AT&T T7102 implements its control/status interface almost entirely in a comprehensive set of 51 8-bit internal registers. Only the transmit-receive buffer description and the buffers themselves are implemented in memory shared with the microprocessor. The 51 internal registers are all directly addressable from the microprocessor by means of a chip select line and bits A1-A6 of the 24-bit address ubs. This bus is normally used for DMA operations.

The 51 registers are divided into five classes:

- COMMAND REGISTER. This register is loaded by the microprocessor to control link initialization, XID password exchange, send/receive enable, and disconnect functions.
- 14 STATUS REGISTERS. These registers are loaded by the controller and examined by the microprocessor. They

provide a comprehensive, real-time picture of what the AT&T T7102 is doing at any given time.

- INTERRUPT REGISTER. Even though the AT&T T7102 provides only a single interrupt request line and no vectoring ability, the reason code loaded into the interrupt register provides a COMPLETE indication of the interrupt cause. With 48 separate codes defined for interrupt register, this is tantamount to having a 48-value interrupt vector. Only one reason code may be presented per interrupt. The simultaneous occurrence of up to five interrupt causing events are accommodated by means of a 4-deep interrupt register FIFO. An interrupt overrun is posted when five interrupts are pending and a sixth interrupt occurs. In no case is an interrupt causing event lost without a trace.

- 17 PARAMETER REGISTERS. These registers are loaded by the microprocessor to control the operation of the AT&T T7102 through system constants (e.g., value of T1, A-field, address of buffer descriptor table, flag fill.).

- 18 COUNTER REGISTERS. These counters make it very simple for the system designer to gather and evaluate maintenance statistics (number of retries, timeouts, REJs, etc.). They are updated by the AT&T T7102 to monitor 16 events on the data link.

The AT&T T7100 uses the same scheme to handle its control/status interface. The only difference is the number of internal 8-bit registers. The T7100 uses 27,

versus 51 for the T7102. The T7100 does not implement internal event counting and therefore does not need the 18 counter registers. Because only modulo-8 is supported, the same functions that require 14 status registers in the T7102 use only eight in the T7100.

3.11) INTERRUPT INTERFACE CHARACTERISTICS

The AT&T T7100 and T7102 both activate a single interrupt request line to start the interrupt procedure. The interrupt request line is deactivated when the microprocessor reads the interrupt register (one of the directly addressable registers).

The T7100 and T7102 provide a single interrupt line. Whenever an interrupt is detected, the reason code is stored in the interrupt register for external microprocessor query. The interrupt code is unique and unambiguous. Should more than one interrupt occur at the same time, or should another interrupting condition occur before a currently pending interrupt has been services, a 4-deep FIFO saves the additional interrupt events until they can be presented.

The unique reason code (27 values for the AT&T T7100, 48 for the T7102) may be used by the microprocessor as a "software vector" in locating an appropriate service routine. Some interrupt conditions are maskable through values set in the parameter registers.

3.12) BUFFER MANAGEMENT

The AT&T T7100 and T7102 manage their send/receive buffers as well as V(R) and V(S) (V(NA) = Next I-frame to be acknowledged, V(S) = Next I-frame to be transmitted), internal state variables by means of buffer descriptor tables in memory. A TLOOK table contains eight descriptor entries, each corresponding to a single transmit frame under modulo-8 sequence numbering. Similarly, an RLOOK descriptor table exists for eight receive frames.

The TLOOK and RLOOK tables are organizes as circular buffers, with V(NA) and V(S) at the head. As I-frames are transmitted or received, the head pointer circulates. This is entirely consistent with HDLC modulo sequencing for I-frames. Each element in the TLOOK and RLOOK tables contains the address of the corresponding buffer, the byte count, and flags indicating the state of the buffer.

Each TLOOK element, for example, contains a buffer ready flag set by the microprocessor and tested by the controller. An acknowledge flag is set by the T7100 and T7102 for testing and by the microprocessor. Together with the V(R) and V(S) counters in the devices' status register, this makes coordination between the T7100 or T7102 and the microprocessor very easy.

To transmit, the microprocessor puts the buffer address and byte count into those in the in TLOOK corresponding to the frames to be transmitted. The buffer ready flags are then

set. After transmission of the previous frame, the next TLOOK entry is examined for a buffer ready flag and transmission of the corresponding frame commences if the flag is set. The T7100 and T7102 step through the TLOOK table circularly until an element with buffer ready flag not set is found.

Meanwhile, I-frame acknowledgements, as they come in, are used to set the acknowledged flags in the TLOOK entries. The acknowledgements also interrupt the microprocessor. Using the V(NA) and V(S) counters, and examining the acknowledged flags, the microprocessor easily determines which frames have been acknowledged, releases their buffers, and updates the TLOOK table with additional I-frames that may be ready to transmit. While this is being handled by the microprocessor, the T7100 and T7102 may still transmit and receive other frames.

Receive operations are handled in a similar way through the RLOOK table. Each entry describes an input buffer by address and size. A buffer ready flag indicates to both the T7100 and T7102 the presence of a buffer. A frame complete flag signals the microprocessor that the controller has filled the buffer.

As each I-frame is correctly received, both the T7100 and T7102 increment their V(R) (next I-frame to be received) counter, update the byte count in the entry, set a frame

complete flag in the corresponding RLOOK entry, and interrupt the microprocessor.

The microprocessor may then unload the received frames by passing their buffer addresses to the packet layer interface, reloading the corresponding RLOOK entries with the addresses of fresh receive buffers, and setting the buffer ready flag again. This allows the receive operation to proceed smoothly and continuously.

To allow some flexibility in system buffer handling, the ST&T T7102 does not require that the send/receive buffers be contiguous (depending on the system, it may be extremely difficult to allocate enough contiguous memory or accommodate a full I-frame). The buffers may be segmented and chained together by means of a chain-address pointer to the next buffer segment. The pointer occupies the last three bytes of each chained buffer except the last one which contains a null pointer. However, since I-frames are typically 256 bytes or less, having a contiguous buffer is usually not a problem.

The T7102 offers either 8 or 128 buffer descriptor entries; 24-bits buffer addressing and two additional descriptors for exchange of XID frames. The T7100 does not provide the buffer chaining feature available in T7102.

3.13) SELF-TEST CAPABILITIES

All four controllers provide provide a loopback feature to allow self-test of at least the data path of the device.

AT&T devices have both an internal and external loopback mode. In internal loopback, the transmitter and receiver are coupled internally. In external mode, the transmit and receive leads must be tied outside the chip. By setting both the receive and transmit station addresses to the same value. The T7100 and T7102 automatically reverse the receive and transmit station addresses when place in near-end or far-end loopback mode to allow the protocol to function properly. The microprocessor must check the transmitted and received data (I-field) for correct loopback operation.

The AT&T devices additionally have an echo mode where all received data is sent back unchanged. In conjunction with a remote station in external loopback mode, this forms a loopback test across the physical link, without having to tie wires.

3.14) NETWORK LAYER SOFTWARE INTERFACE

Devices of the size and complexity of these two controllers require significant amounts of microprocessor software control. Initialization, interrupt handling, data buffer management, and exception condition handling are such functions.

This means that the controllers, by themselves, do not constitute a complete OSI data link layer that can be driven from a network layer process. The device must be

combined with its supporting handlers to form a complete data link layer entity.

CHAPTER FOUR

THE COMMUNICATION BOARD SYSTEM OVERVIEW

Section 3 & 4 will introduce the hardware for this system. The system software used in this communication board will then be explained in sections 5 & 6.

First the heart of the system the CPU. The CPU used in this system is the Motorola MC68000 with a 16bit data bus and a 24bit address bus. This particular CPU was chosen because we had a ROM monitor to run the MC68000 CPU and all the hooks were in this monitor for C's input and output facilities. Therefore it was not a major task to implement "C"'s I/O.

A "C" cross-compiler was provide for our host UNIX system. This C cross-compiler and library of System Calls were provide on a magnetic tape from another UNIX machine. It was required to install this C cross-compiler on our host UNIX system and develop our system interface needed for this project. The system interfaces are the proper directory structures, C utilities and library routines. A MAKEFILE helped provided the correct system interfaces for the C cross-compiler inherited path structure and system interface. This MAKEFILE also created the C compiler for system. In addition to the C compiler we needed to install the library routines and the other C utilities.

The X.25 communication is provided by AT&T's T7100 and T7102 protocol controllers(the XPC's). The XPC's performs

complete link level control according to the X.25 data communication protocol. It generates supervisory and unnumbered frames automatically without intervention of the host CPU. The Host CPU must program the XPC's, and supply buffers for the data fields of the received and transmitted information frames. The CPU is notified of important events via XPC interrupts. The XPC's contain a transmitter, a receiver, an XPC controller and an unit interface.

There is 28K of static RAM provided. In addition there are hooks to provide 1MEG of dynamic Ram. All the devices that need to be written to or read by the CPU were memory mapped by the address decoder section of the communication board. Some of the devices that fall into this category are the 9519(Interrupt Controller), 8255(Peripheral Interface controller), DUARTS(I/O PORTS), and the XPC's.

4.1) THE COMMUNICATION BOARD LAYOUT

This section will explain FIGURE 2. FIGURE 2 is the communication board hardware layout. This is a block view of the major devices involved with the communication board. In this section we will show the basic interaction between these devices and there dependencies on one another. The system is driven by the MC68000 CPU and the main communication between this CPU and the devices on the system is the address bus(ADDBUS) and data bus(DATABUS). The CPU can address any device through these buses. The

information that the CPU provides on the bus is latched through the address buffers(BA) and the data buffers(BD). This latching ensures that the data will be available for the devices to read/write and that the ttl signal levels are correct. The second set of buffers BA1/BD1 provide the same service for the XPC's when they are driving the bus to access RAM memory in the DMA mode.

The dotted frame represents the area which will provide the hardware hooks for 1MEGABYTE of dynamic RAM.

The ROM monitor is represented by the EPROM. This is the resident communication board system software which drives CPU.

The 28k of static RAM is represented by SRAM. SRAM is the area of memory area provide for downloading and executing programs and for memory storage locations of uninitialized ROM data.

Other devices are in the layout are the XPC's, which give us the X.25 LAPB level-2 point to point link with another computer via its RS232 interface. There are four RS232 links for X.25 communication.

The 9519 is an interrupt controller. This controller schedules and prioritizes the interrupts coming from the XPC's. These are then reported to the CPU. The CPU will then, in turn, service that device.

The 8255 is a programmable interface controller. This device has two purposes. The first purpose is to provide

clocks to the XPC's. These clocks are programmable for various baud rates on the XPC. Also this devices will allow us to reset any of the XPC devices.
The section which decodes the address and chip selects the correct device is called the address decoder. This address decoder is made up of several logic decoder functions. These functions provide an address for all needed devices. The duarts transmit and receive information from either a terminal or the host UNIX system via their RS232 link.

4.2) THE MOTOROLA 68000

The Motorola 68000 is the CPU (or sometimes referred to as the MPU) for our communication board. This requires that all the hardware be designed with its lead interfaces in mind. Also the software and hardware must be designed with the CPU's memory byte orientation structure in mind. This CPU's design for memory addressing is simple yet powerful. There are no page references to memory like intel's processors. This allows us the freedom in programming of not having to worry if data section and program section of executable code will fit in certain areas of memory confined to a certain page.
Another advantage of the Motorola 68000 is that it sees everything as a memory location. This allows us to memory map all the hardware into memory address locations. What this means to the programmer is that a programmable device in memory can be represented as a data structure. Since

the devices can be represented as data structures one need not resort to programming the devices in assembly code to access registers on these devices. All the software for initialization devices that will not reside on the monitor can be written in C.

Since a large part of the software written requires addressing devices as data structures an example would be in order. The way to point to arbitrary locations in memory using C is by means of explicit conversions. Using this type conversion forces the data structures to a certain area of memory.

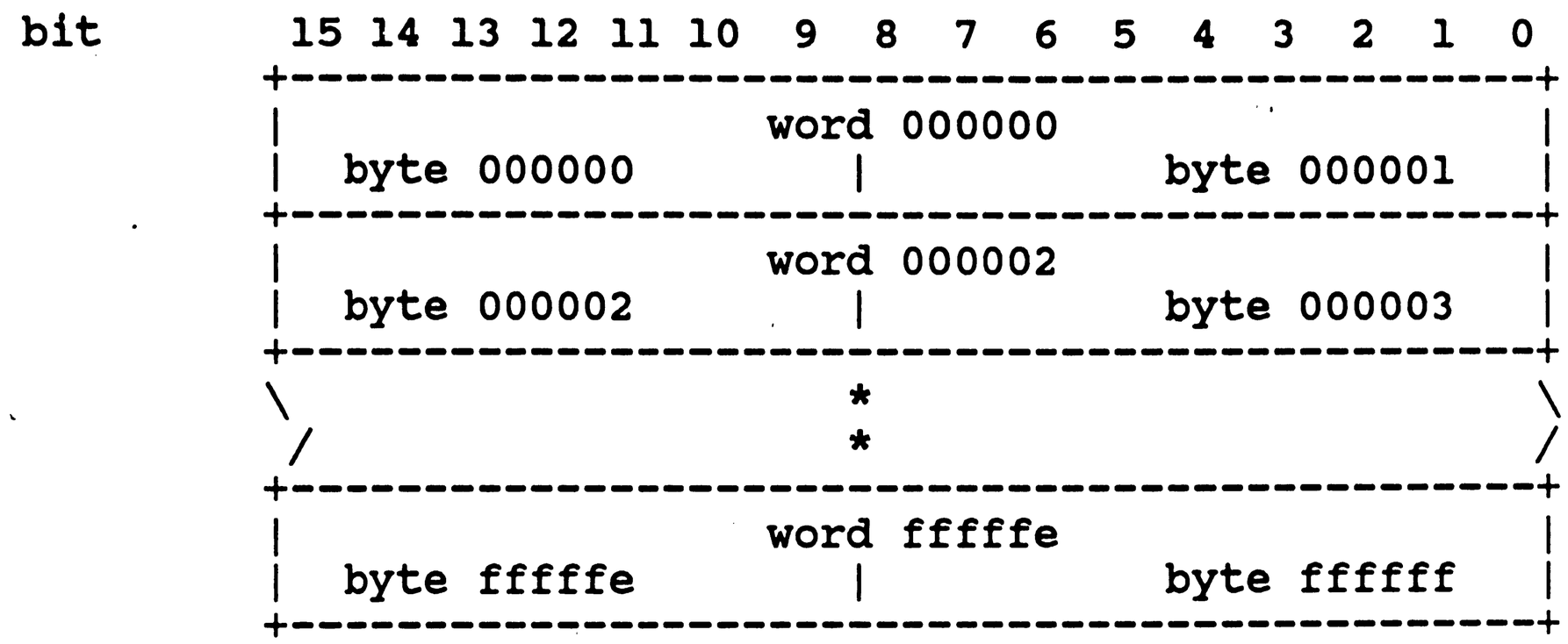
EXAMPLE "C" CODE:

```
/* PROGRAMMING THE ADM9519 INTERRUPT CONTROLLER */
/* point to 9519 at address 0x214000 */
#define USR_9519    ((struct intcntl *) 0x214000)

     /*
      *                                          7      0
      *                                         +--------+
      *   (struct intcntl *)->data              |        |
      *                                         +--------+
      *
      *                                          7      0
      *                                         +--------+
      *   (struct intcntl *)->control           |        |
      *                                         +--------+
      *
      */

struct  incntl
     {
          /* offset for 16bit word */
        unsigned :8;
          /* 8bit data register */
        unsigned char data;
          /* offset for 16bit word */
        unsigned :8;
          /* 8bit command register */
        unsigned char command;
     };
/*   Thus the AMD can be written to either of the following
                    ways:  */
USR_9519->command = 0x04;
/* OR */
((struct intcntl *) 0x214000)->command = 0x04;
```

When accessing devices in memory or trying to access certain bits it is important to realize where these bits are physically located in memory and what the bit order is in these memory locations. The CPU has a 16bit data word but is also capable of having 32bit data representations. The word organization is as follows:



This information is crucial when trying to access a certain bit in memory. So this information guides the C structure as to which bit to set.

For instance the XPC's require the CPU to flip certain bits in memory for memory transfer verification. This is done in ROM memory locations of the XPC's Transmitter/Receiver Look-up tables.

An example of this is in the function *init_table()* where we initialize and set bits in RAM memory for the XPC's TLOOK elements. We've defined :

tables[table]->rlook_element[i].brdy = DISABLE

from typedefs.h the structure *TLOOK*. Each of those first four elements in *TLOOK* is defined as one bit. Thus the order of the bits and bytes is required for the programmer if setting bits is required in the program.

The hardware implementation also needs this information. The particular device that requires memory access is the XPC's. The XPC's do memory access via DMA transfers from

the chip to memory. Since these are 8bit transfers into memory and the memory locations are set up for 16bits, the hardware needs to map the correct information to the correct memory location. An explanation of this will be found under the XPC section.

4.3) THE MOTOROLA 68000 MONITOR

The CPU monitor is the software which can be considered a miniature operating system. It supplies us with the needed interface into the communication board and the communication with the host UNIX system.

The 68000 CPU ROM monitor provides us with the basic interface into the communication board . The monitor is a set of software instruction which control the communication board. This is a miniature operating system located in 4K of on board ROM. It will provide us with a set of commands that allow us to work with the communication board. These monitor commands allow communication between a terminal and our communication board, there is also a communication link between the communication board and the host UNIX system.

The communication link between the communication board and the host UNIX system allows us to directly talk to the UNIX system without interference from the communication board. This communication board UNIX link also allows us to download the compiled 68000 executable code from the UNIX system. The executable code was compiled from our C cross-compiler residing on the host UNIX machine. Executable

code that is downloaded from UNIX can then be executed by our ROM monitor using the *GO 0xADDRESS* command where 0xADDRESS is the starting address of the executable code. The monitor will run this code and provide the basic exception processing and recovery. Code which is in RAM and changed (or developed there) can be up loaded to UNIX for further processing if necessary.

In addition to these communication interfaces the monitor will allow us to assembly/disassemble 68000 assembly code directly in RAM. Write/read areas of RAM and ROM. Also the are provisions for moving blocks of data in RAM.

Some of the rudimentary initialization function provided by the monitor are for the duarts, interrupt vector functions and "C" I/O links.

We provided the initialization code in the monitor for the duarts. Also we adjusted the monitor for keeping track of the putc and getc routines links with the C programs. This involved understanding the duart and its functionality and how the C compiler works.

4.2.1) EXCEPTION PROCESSING

The CPU provides us with exception processing. Exceptions are generated by either internal or external causes such as interrupts, divide by zero, BUS error, excetera. The monitor provides for the internal exceptions processing, while we provided the exception processing for the external events such as interrupt service routines. These service

routines were provided by inserting the correct address of the service routine in low memory rom. Low memory rom has all the vector addresses for the exception vectors known as the exception vector table.

The way the CPU responses to an interrupt or other exception is in four steps. The first step is to make an internal copy of the status register, the assert the S bit, putting the CPU in supervisor state, and negate the T bit, to turn off tracing. In the second state the vector number of the exception is determined. The vector number is then used to access the vector number table to generate the address of the of the exception vector. The third step is stack maintenance and the last step is to get a new program counter from the exception vector and execute the service routine.

So by providing a address in the exception vector table we can put service routines links into the system.

Our communication board provides exception processing for the XPC interrupt routines. These are services which help keep the X.25 link functioning properly.

4.4) THE DUARTS

The Motorola 68681 dual universal asynchronous receiver/transmitter (DUART) is part of the 68000 family of peripheral devices and directly interfaces to the MC68000 processor via an asynchronous bus structure.

The function of the DUARTS in our system are for I/O. There are two duarts in the system and each DUART has two channels. Each channel of the duarts is through an RS232 link. One DUART has the channels for a terminal and the other channel is for the modem interface with the UNIX host link. The second Duart has the capabilities for either two more terminals or a printer and a terminal. These channels are set up for 9600 baud full duplex. This initialization is done in the CPU monitor, which addresses the DUARTS and sets the proper registers.

Here is how the DUARTS interact with the software is when a character is received through the RS232 link. Once the DUARTS receive a character from its channel it will issue an interrupt to the CPU. The CPU in turn will provide the proper exception handling and respond with the service routine which reads the DUART and gets the character for further processing by the monitor. The DUART can output a character to a channel by having the CPU write the character to the DUART then the DUART will transmit the character across the channel complying with the proper RS232 interface.

4.5) AMD 9515 PROGRAMMABLE INTERRUPT CONTROLLER

The AMD9519 Universal interrupt Controller is a processor support device designed to enhance the interrupt handling capabilities of our 68000 system. The 9519 manages the masking, priority resolution and vectoring of our 4 XPC

interrupt requests. The 9519 responds to the host CPU during a interrupt acknowledgement process by providing the vector address for a particular interrupt.

4.6) INTEL 8255

The Intel 8255 is a general purpose programmable I/O device. It has 24 I/O pins which may be programmed in 2 groups of 12 and used in 3 major modes of operation.
In our system the 8255 is used to control the clock rate of the TC/RC (Transmit/Receive clock) on the XPC's RS232 level-1 interface. The PA and PB pins on the 8255 are use to drive LS151's. These LS151(1 of 8 data selectors) then selects the proper baud rate from a LS393 four bit counter with two clock inputs. This counter provides baud rates from 1200 baud through 15.2k baud.
The remainder of the PC pins on the 8255 are used to reset any of the XPC's. This is done by activating the Master Reset on the particular device.

4.7) THE XPC DIRECT MEMORY ACCESS DMA CIRCUITRY

The schematic for this circuit is on sheet 3. This circuit allows the XPC's to request control of the system bus from the CPU. The CPU will then allow the particular device to be a bus master for a given XPC DMA cycle period.
The request for the bus from the XPC's come into U65 an ls373 latch. These signals are then passed into U59 an 8 to 3 priority encoder. This will then generate the proper DREQ (DMA request) and triggers the proper DMA acknowledge

(DACK) once *DGRANT goes active. This circuitry also takes care of the keeping the *BGRANT active to keep other activity off the bus until the XPC's DMA cycle is through.

4.8) ADDRESS DECODER CIRCUITRY

The address decoder circuitry decodes the addresses produced by the CPU and provides the Chip Select to that particular device whether its a memory device or another type of device. This decoder will also provide a DTACK back to the CPU to indicate that data has been acknowledge and received by the device.

U6 will decode in the range of <= 0x2fffff. U8 will provide an *ACK for anything in the range of 0x300000 to 0x800000 with an *ACK.

These *ACK will go through U12(ls 175) for a clock delay response of *DTACK. However this is ored with *ACK via U68 (ls32 p2). The idea here is to provide a *DTACK response so not to hang up the CPU, when addressing an area of memory which is out of bounds or a devices which will not provide a *DTACK response (EPROMS, STATIC RAM). The purpose oring the with *ACK is to drop the *DTACK if it has already been provided by a device.

4.9) ADDRESS BUS CONTROL

In a system such as this, one must control the direction of various buses and tri-state those that are not currently in use. This section will explain the address bus and how the flow of information is directed on this bus. In our case

the address buffers are numbered from one through six and the data buffers are numbered from one to five.

If a device is not on the bus, then it is expected that the buffer will be TRI-STATED. These are the conditions to design for:

1) The CPU is reading or writing to peripheral devices.

2) The CPU is reading or writing memory.

3) The peripheral devices are reading or writing memory.

The state table is given below:

	ADDRESS BUFFERS					
	1	2	3	4	5	6
CPU to PERIPHERALS	E	E	E	E	E	D
	E	E	E	E	D	E
CPU to MEMORY	E	E	E	D	D	D
XPC TO MEMORY(DMA)	D	D	D	E	E	E

D = Disable
E = Enabled

The R/W signal ia a directional signal which is used by both the CPU and the XPC peripheral device. Only buffer #4 is bidirectional because the CPU must address the registers of the XPC devices and the XPC are located on an even address leads.

	ADDRESS BUFFERS					
	1	2	3	4	5	6
CPU R	O	O	O	I	I	I
CPU W	O	O	O	I	I	I
CPU to MEMORY	O	O	O	D	D	D
XPC TO MEMORY(DMA)	D	D	D	O	O	O

D = Disable
O = Output
I = Input

Because the direction of some buffers is governed by more than one peripheral, the control signal is placed on one of the control bus leads after being altered by the appropriate logic.

So to provide the proper enable signals to the address buffers, we use the state table above to come to the following logic.

Page 2 U1, U2, U3 can be enabled whenever the CPU wants to wants to read or write devices(including memory) and the BGACK signal will indicates that the CPU has the bus. So the flow of information will always be from the CPU side to the address bus.

U32 enables the direction of the address bus to and from the XPC's. If the CPU has the bus then *BGACK will be inactive and BGACK will be active and the flow of information will be from the CPU into the XPC's. Now when the *BGACK signal is active the flow of information will be from the XPC's to memory, this is when the XPC's are in DMA mode trying to access memory. U36 uses signal b36e in the same fashion as U32 uses BGACK however the signals come from the T7102 which has a larger address bus

U31 is a little different. It is controlled by the signal ACSX. ACSX indicates that one of the XPC's have been chip selected. So this will activate(output enable) meaning that information will flow through U31. Now the direction of the information will depend on whether the CPU is

writing/reading the XPC registers, or if the XPC is in DMA mode trying to access memory. The U31 direction on pin 1 will depend on who has the address bus.

CHAPTER FIVE

THE XPC'S

The XPC's provide us with the complete level-2 LAPB protocol according to the X.25 data communication protocol. It generates supervisory and unnumbered frames automatically without intervention by the host CPU. The host CPU must program the XPC's, and supply buffers for the data fields of the received and transmitted information frames. The CPU is notified of important events via the interrupts. The XPC's contain a transmitter, a receiver, an XPC controller and an interface unit.

5.1) THE XPC'S RS232 INTERFACE

The XPC's interface into level-1 of X.25 via RS232. The signals needed for RS232 are provided on the chip as ttl level signals. These signals are then converted from ttl level to the proper RS232 signal level (+/- 12 volts) by means of two chips an ls1488 and ls 1489. The RS232 signals are provided through the transmit and receiver units on the XPC's. The signals provided by the XPC's are the following signals.

Transmit Data (TD) the serial data output line.

Transmit clock (*TC) 1x clock input needed to operate the transmitter. Received data (RD) XPC serial data input lead.

Receive clock (*RC) 1x clock input controlling the receiver. Carrier Detect. Indicates level-1 interface is

receiving and modulating a usable signal. Clear-to-Send From level-1 interface indicates to the XPC that the full duplex link is ready. Request-to-Send indicates that the XPC is requesting the physical link.

5.2) THE XPC'S CPU INTERFACE

The XPC provides the system with the interface to transfer information from the host CPU system to another X.25 LAPB system over the level-1 RS232 physical link. The interface unit on the chip provides the interface between the host CPU and the XPC transmitter and receiver via triple channel DMA. The CPU/DMA memory interface will be explained in the software section of this paper.

5.3) XPC'S BUS AND I/O LOGIC

The master reset in the XPC-8 in addition to resetting the device is used to select between the Intel and Motorola bus control protocols. A valid reset request pulse must be held low for at least six cycles of CKO (buffered internal clock, which is half the frequency of the input clock). The glitch protection circuitry guarantees that glitches of less than one CLK (master clock input) period are ignored. If a single valid request is provided, the XPC-8 bus will be configured for the Motorola bus mode. If a second valid request is provided within 30 cycles of CKO, the XPC-8 switches to the Intel bus mode. The MR pin must be held high for at least six cycles of CKO between reset pulses for the second reset pulse to be recognized the the XPC-8.

Any reset pulses that occur after the 30 cycles of CKO are interpreted as the beginning of a new reset sequence. Either the Motorola or Intel bus modes can then be selected.

We can select the Motorola mode in our system in two ways. First is to use the hardware reset button. Second is to use the software reset by setting a bit in the intel 8255 as explained earlier.

Bits A0-A5 of the address bus are bidirectional and are used to access the internal registers of the XPC'S. When the chip is selected, R/*W, *WE, *DSRE, and the address lines A0-A5 become inputs. *READY becomes an output. In our system *READY is referred to as a DTACK signal to keep with the motorola bus reference since it serves the same purpose.

DMA operations are controlled by the Interface Unit on the chip. The XPC uses DMA to main memory to read and write TLOOK tables elements, RLOOK table elements and data fields of information frames.

5.4) XPC'S PERIPHERAL MODES

When addressing the XPC on chip registers set, four leads are necessary, in addition to the address and data leads. The four leads necessary are the *CS, *DSRE, R/*W and *READY. THe *CS or chip select lead is obtained from our address decoder circuitry. R/*W is tied to our R/*W bus which is driven by the R/*W of the CPU which is tri-stated

when the XPC is in the DMA mode. *READY is used as *DTACK or data transfer acknowledge. The *DSRE which is used as a data strobe in our system is generated by the address decoder chip select. and a tri-stateable buffer such as the LS125. The LS125 provides isolation for the *DSRE pin and also provides us with a switch to turn on the *DSRE signal during chip elect to ensure the data gets strobed in during the correct timing. Another reason to use the buffer is because the *CS signal is not tri-stated during XPC DMA cycles. Figure 3 shows in isolation how it is done in our system.

Reads or writes to XPC registers take a minimum of four CKO (pin47 XPC-8) clock cycles depending on when *CS and *DSRE become active with respect to the CKO clock. The *READY lead signals the CPU to terminate the current cycle.

5.5) DMA

When the XPC issues a DMA request. It causes the *DREQ (DMA request) line to become active and wait for the CPU to issue a DTACK which tells the XPC that it can have the bus. Section 3.7 explains how the communication board handles the problem with multiple XPC DMA's. At this time six leads in addition to the address and data lines must be considered. These are *PARV, *DSRE, *WE, *READY, R/*W, and *AS. In our system we are not using parity checks into DMA memory transfers so we tie *PARV low. The other leads are

standard in operation. They behave as if the XPC were the CPU.

When the XPC places the a valid address on the bus , the *AS is active, then when the XPC places valid data on the the data bus the XPC causes causes the *DSRE to become active iin case of a read and *WE to become active in the case of the write. The XPC expects to receive a signal on the *READY lead to indicate that the data read/write cycle is terminated. *BGACK is the DMA request lead generated to indicate that the XPC's are driving the buses. When it is active the XPC's are bus masters.

5.5) DATA BUS INTERFACE

Because we are interfacing the eight bit data leads of the XPC to a 16 bit data bus we must be able to address each even or odd memory location. Figure 4 shows the interface circuitry from the data bus DB0-DB15 to the XPC data leads XD0-XD7. The only signals we are not familiar with are B3E, B4E, and BUFDIR. When the CPU is trying to read or write to the XPC registers, either B3E or B4E should be enabled, the direction must be from B to A when the CPU is reading and from A to B when the CPU is writing. When the XPC is bus master, the direction is from A to B when reading and B to A when writing. When there is no activity concerning the XPC, those buffers are tri-stated.

From figure 4 one can confirm the above results. If *CS and *BGACK are both high, both buffers are tristated so

direction does not really matter. If XPC *CS lead is active then the CPU R/*W line is active and the direction depends on whether it is reading or writing.

CHAPTER SIX

SYSTEM SOFTWARE

This section will examine the system software of the communication board. The software can be broken down into a few distinct blocks. First is the CPU monitor which runs the 68000. Second is the initialization and housekeeping code for the devices on the board. Last is the MAKEFILE which keeps everything in order.

First thing that will be done in this section is elaborate a little more on the CPU monitor and the C interface (compiler, mxhex, etc). Then we will get into initialization of the devices and explain in detail how to set up the parameters on the XPC-8. Finally in trying to keep sanity in code development we will examine our MAKEFILE and how it supports software development.

6.1) THE CPU MONITOR

In section 3.3 we discussed what the CPU monitor is and a little bit of the workings of the monitor. Now we will go into detail of what we needed to implement in the monitor to make it compatible with our system.

The first thing that we needed to do was to section memory according to functionality. We have set aside address location for all the devices on the communication board this also includes the EPROMS and RAM memory. So the EPROMS are located at address 0x0000 to 0x4000 and RAM memory is from 0x4000 through 0xffff and 0x20000 through

0x2ffff (0x defines a hex number). Locations 0x4000 through 0xffff is set aside for variable locations of the CPU monitor and XPC buffer locations. While 0x20000 through 0x2ffff is set aside for programs. As stated earlier low ROM 0x0000 through 0x03ff are for the exception vector table.

On the first page of monitor code is the vectors we use in ROM. These are commented out because we had to program the low ROM by hand and these are there just to remind us. The Motorola 68000 does not have an address pin of A0. The DUARTS are memory mapped at 0x200000 and have four address pins A0 through A4. Consequently the addresses of the DUARTS are on odd boundaries.

Most of the work involving the monitor was debugging the monitor for our hardware and software. The hardware debugging involved making sure all the addresses were indeed correct and the CPU monitor initialization code did what we though it was suppose to do. We also had some problems which involved memory overlap. Since we were just beginning the system it was a matter of keeping a map of the location and segmenting it accordingly.

6.2) THE C COMPILER

The C compiler was given to us on a magnetic tape which was archived from another UNIX system. The compiler was installed through a series of MAKEFILES which, after modifying these MAKEFILES for our own UNIX path structure,

the MAKEFILES set up all the proper C utilities in the correct places.

The tools provided to by this C compiler were of course the C compiler itself. This C compiler provided us with the option of starting the code at a certain address location in memory. This allows us to keep our programs confined to certain sections of memory in accordance with the maps and allocation described in the above sections.

The other tools provides were a hex format program called mxhex. This program takes our C executable binary code and formats it as a intel format hex dump. This provides us with code that can be down loaded into our communication board and verified in the download by the CPU monitor.

6.3) THE INITIALIZATION CODE

The file *typedefs.h* defines all the devices and memory buffers in the system as data structures. These data structures then access the registers on the devices themselves. The devices are then programmed according to the system requirements.

The data structures we defined in *typedefs.h* are tied in directly with the hardware. The registers on all the devices in the system are 8-bits and the address bus is 16-bits. Some places the data structures need some offset fill to make sure they land on the right word boundary. Two examples of these offsets are the AMD9519 and the Intel 8255. Both of these devices have their A0 address leads

connected to the A1 lead of the address bus. Therefore the registers are addressed on odd address boundaries. There needs to be some offset in the data structure to force the data to the odd register of the 16-bit data bus. This is done by defining everything as a unsigned character. First an offset of 8-bits defined as unsigned characters defines the even part of the 16-bit word, then the data register defined as a unsigned char defines the odd part of the 16-bit word. This is shown in an example in section 3.2. The XPC's are set up to be on even addresses. These are not required to have offsets.

6.4) SETTING UP THE XPC'S SYSTEM REQUIREMENTS

The system requirements are defined in the XPC's for one session. A session is the bringing up of a link and the the transfer of the required information and finally the closing down of the link. The session is set up by writing the parameter registers with certain parameters for this particular session.

Explained in detail will be the T7100 (XPC-8). This device was our workhorse there were three XPC-8's on the communication board. They were tested fully with positive results. Also the T7102 has its sessions setup in a very similar manner and most of the parameters registers are the same and the values(parameters) set in these registers are the same. So we will explain all the requirements of LAPB

needed for initializing the XPC-8's and what to program into there registers.

6.4.1) MASTER RESET

The hardware must reset the XPC-8 by activating the MR pin at system startup. This puts the XPC-8 in a setup state where the device is non-operational; i.e., it is not engaged in X.25 data communication. In this setup state, the XPC-8 can be configured for a specific application by writing the parameter registers. We have two means to master reset on the communication board. First way is a button which provides a hardware reset and the second way is a software reset by toggling the Intel 8255 ports. All outputs are put in the high impedance state when the master reset MR is asserted. The parameter registers are then set to zero and the command register is set to 82H (DISCMOD = 1, MDISC = 1, all other bits are set to zero). A one(1) in the mandatory disconnect (MDISC) field of the command register causes the logical link to go to a logically disconnected state. The XPC-8 responds to all inquiries with a DM (disconnected mode) frame when it is in this state. A zero(0) in the MDISC field allows the XPC-8 to establish a logical link in the operational state. The disconnect mode (DISCMOD) bit in the command register (bit 7) specifies which of the two disconnected states the XPC-8 assumes when it is logically disconnected. A one(1) in the DISCMOD field permits the XPC-8 to respond with a DM

frame only to command frames with the P-bit set. The XPC-8 is in a "super" disconnected state.

6.5) XPC-8 REGISTERS

Internal accessible registers of the XPC-8 are mapped into the system memory space by the hardware and they are represented as data structures in the C code. Address lines A5 - A0 become chip inputs to address the internal registers when chip select (CS) is inserted.

6.5.1) COMMAND REGISTER

The 8-bit command register is used to control seven XPC-8 functions. They are as follows:

1. Send Permission (SEND): Controls the transmission of data by inhibiting or enabling the sending of information frames (I frames).

2. Receive Ready (RECR): Indicates that empty receive data buffers have been allocated and are available.

3. Active/Passive Link Initialization (ACT/PAS): Specifies if the XPC-8 will actively initiate a link setup (i.e., send a SABM frame) or if it should passively await a link setup.

4. Password Exchange (PWXCH): Specifies whether the XPC-8 will initiate a password exchange procedure prior to link setup.

5. Password Verified (PWOK1 & PWOK2): These bits notify the XPC-8 of the password exchange verification.

6. Disconnect Mode (DISCMOD): Specifies which of two disconnected states the XPC-8 assumes when logically disconnected.

7. Mandatory Disconnect (MDISC): Causes the XPC-8 to go into a disconnected state. While in the "super" disconnected state, as determined by DISCMOD, the XPC-8 responds with a DM frame only to frames with the P-bit set.

6.5.2) WRITING PARAMETER REGISTERS

The XPC-8 is configured by writing the parameter registers when it is in the setup state. The MDISC bit in the command register is cleared after the parameter registers are written, putting the XPC-8 into a operational state. If the Active/Passive bit is set in the command register, clearing MDISC will initiate link setup; i.e., and SABM frame will be transmitted.

6.5.3) PARAMETER REGISTERS

There are 17 parameter registers in the XPC-8 that specify system constants, link parameters, and modes of operation. Once the XPC-8 has been placed in the operational state, it can be reconfigured only after a reset. The specific values given for the parameter registers are systems parameters which were agreed to for a period of time with the DXE.

Parameter register zero specifies link tests. During normal operation, each of these bits have the value zero. Only one of these tests can be activated at any time.

REG 0 BIT 0 NEAR-END LOOP TEST

Setting this bit causes the XPC-8 to perform the near-end loop test. All essential X.25 parameters should be specified before entering this mode of operation. In this mode, the internal receive data signal and internal transmit data signal are tied together. The receiver automatically interchanges the command and response addresses to allow the protocol to function properly. The TD pin remains high while in this mode.

REG 0 BIT 1 FAR-END LOOP TEST

Setting this bit causes the XPC-8 to perform the far-end loop test. All essential X.25 parameters should be specified before entering this mode of operation. The far end of the link should be in an echo mode during this test. The far-end loop back permits the XPC-8 to talk to itself through the underlying physical layer. Data transmitted by the XPC-8 will be passed to the physical layer for data transmission over the physical medium. Signals received at the remote physical layer will be processed and passed to the data link layer. Data arriving at the remote XPC-8 will be internally routed, without CPU of XPC-8 intervention, to the transmit data output for transmission by the physical layer over the physical medium. The receiver in the local XPC-8 will interchange the command and response addresses to allow the protocol to function properly.

REG 0 BIT 2 ECHO MODE

Setting this bit causes the XPC-8 to transmit what it receives unaltered.

REG 1 BITS 0 - 2 WINDOW SIZE, K

The window size k is the maximum number of outstanding (i.e., unacknowledged) I frames permitted at any given time. It can be any number from 1 through 7. Reducing the number of outstanding frames allows optimization of buffer usage among different users. However, the efficiency of link utilization increases with a larger k value because less time is devoted to waiting for acknowledgments. Factors influencing the choice of window size included the loop transit delay, memory resources, and link utilization.

REG 1 BIT 4 READY BUT IDLE

This bit is used only when the logical link is disconnected and MDISC = 0. If this bit is a 0, the XPC-8 sends flags between the frames. If it is a 1, the XPC-8 idles (sends 1s) between the frames. Continuous 1s are not sent in the link setup or information transfer states. The XPC-8 will utilize this bit only when it is in passive mode and waiting for an SABM. The transmission of flags between frames is required in normal X.25 operation.

REG 1 BIT 5 XID ENABLE

Setting this bit enables the password exchange (PWXCH) mechanism. A successful exchange of passwords is then a prerequisite for link setup.

REG 2 - 3 BITS 0 - 7 TLOOK STARTING ADDRESS

The TLOOK starting address is the memory location of the first element in the transmitter and receiver lookup tables. This should be a block of main memory 128 bytes deep (144 bytes if using password exchange) to accommodate the eight 8-byte elements of the TLOOK and eight 8-byte elements of the RLOOK tables (password exchange uses an additional 16 bytes). This address must not be zero. As a safety feature, if the TLOOK starting address is inadvertently set equal to zero, an interrupt is issued, and the XPC-8 halts operation.

REG 4 BITS 0 - 7 PARAMETER T1 (LOW BYTE)

REG 5 BITS 0 - 3 PARAMETER T1 (HIGH BITS)

The period of T1 is the time limit set for the primary acknowledgment timer. It is the maximum time that the XPC-8 waits for an acknowledgment of previously transmitted I frames or poll bit frames. The hardware timer in the XPC-8 is a counter that counts cycles of the system clock CKO prescaled by the factor 2^{15} = 32,768. The value of the primary link acknowledgment timer T1 is specified to the XPC-8 as a number of prescaled clock counts. It is calculated as follows: period of T1 = 32,768 * (T1 parameter/fCKO)

The T1 timer is started at the end of the transmission of I frames and poll-bit frames. The T1 value should be selected to allow for the transmission of the frame across

the physical medium, the processing of the frame on the remote side, and the generation and transmission of an appropriate response (e.g. RR, RNR, or I frame) which acknowledges the frame.

Once the period of T1 is defined, the T1 parameter can be calculated by: T1 parameter = (fCKO/32,768) * (period of T1 timer)

Registers 4 & 5 store the T1 parameter, the number of counts of the scaled system clock that provide the desired period for the acknowledgment timer.

REG 5 BITS 4 - 7 N2 X.25 RETRANSMISSION COUNTER

The value N2 is the maximum number times that a frame is transmitted without receiving a response. If, after N2 attempts to transmit a frame, no response is received, it is assumed that the link is dead and the XPC-8 is placed in a disconnected state.

REG 6 BITS 0 - 7 PARAMETER T2 (LOW BYTE)

REG 7 BITS 0 - 3 PARAMETER T2 (HIGH BITS)

The period of T2 is the maximum time that the XPC-8 waits before responding to an inquiry from the remote DXE. The period of the T2 timer is a function of CKO and the T2 parameter.

When the period of T2 is defined, the value of the T2 parameter is determined by T2 parameter = (fCKO/32,768) * (period of T2 timer)

REG 8 BITS 0 - 7 N1 (LOW BYTE)

REG 9 BITS 0 - 4 N1 (HIGH BITS)

N1 is the maximum number of bytes (up to 4096 bytes + L3 header) in the information field of an I frame. Receiver buffers must be large enough to hold I fields that N1 bytes long.

REG 10 BITS 0 - 7 PARAMETER T4 (LOW BYTE)

REG 11 BITS 0 - 3 PARAMETER T4 (HIGH BITS)

T4 is the maximum time that the XPC-8 will go without exchanging frames on the logical link. The purpose of the T4 timer is to detect link level malfunctions. During periods of link inactivity, the XPC-8 will query the remote DXE for its status every T4 seconds. The XPC-8 will go to a disconnected state if the remote DXE does not respond satisfactorily to a link status query after N2 attempts. If it is necessary to detect link level errors quickly, T4 should be set to a relatively small value, but T4 should always be greater than T1. When the period of T4 is defined, the value of the T4 parameter is determined by:

T4 parameter = (fCKO/32,768) * (period of T4 timer)

REG 12 BITS 0 - 7 PARAMETER T3 (LOW BYTE)

REG 13 BITS 0 - 3 PARAMETER T3 (HIGH BITS)

T3 specifies the amount of time that the XPC-8 accepts receipt of an idle condition before taking alternative action. There is a physical level malfunction and the remote DXE is not operational (i.e., transmitting all 1s) when the idle link timer expires. If it is important to

detect physical malfunctions as soon as possible, T3 should be a small value. If rapid detection is not needed, the T3 can be a larger value in order to give the physical level maximum time to recover. The period of the T3 timer is a function of CKO and the T3 parameter.

The value of the T3 parameter is determined by: T3 parameter = (fCKO/32,768) * (period of T3 timer)

REG 14 BITS 0 - 7 TRANSMIT COMMAND ADDRESS

This register contains the data link layer address of the remote DXE, i.e., the station at the other end of the link. If the XPC-8 is used as a DTE in a network application, this register contains the DCE address 1 (00000001).

REG 15 BITS 0 - 7 TRANSMIT RESPONSE ADDRESS

This register contains the data link layer address of the local DXE, i.e., the address of this station. If the XPC-8 is a DTE in a network application, this register contains the DTE address 3 (00000011).

REG 16 BITS 0 - 7 FLAG COUNT

Specifies the minimum number of extra flags between frames. The value of flag count is cleared (set to zero) on reset, which causes the minimum number of flags between frames to default to 1. This feature is used to slow down the rate of frame transmission by the XPC-8, thereby permitting slower (software-based) implementations to process a frame before the next frame arrives. No extra flags are needed

when communicating with another XPC-8; use the default of zero.

XPC-8 INTERFACE TO SYSTEM MEMORY

System memory is where the data buffers that store the transmit and receive data are located. The transmit and receive data buffers are pointed to an defined by the transmitter lookup table (TLOOK) and the receiver lookup table (RLOOK), respectively (see Figure 5).

TRANSMITTER LOOK TABLE

The TLOOK table is a block of eight 8-byte elements in system memory. The base address of the TLOOK table is specified by the TLOOK START ADDRESS which is contained in parameter registers 2 and 3. Each element in the TLOOK table describes a buffer corresponding to one packet of data to be transmitted by the XPC-8. The TLOOK table is maintained as a modulo 8 circular queue. Figure 6 shows the structure of a TLOOK element. Table 1 describes the bit field in each TLOOK element.

RECEIVER LOOK TABLE

The RLOOK table is also a block of eight 8-byte elements in system memory which immediately follows the TLOOK table. Each element in the RLOOK table describes a buffer which holds one packet of data received by the XPC-8. The RLOOK table is maintained as a modulo 8 circular queue. Figure 7 shows the structure of a RLOOK element and Table 2 describes the bit field in each RLOOK element.

TRANSMITTER PASSWORD EXCHANGE LOOKUP TABLE

The TXID table is a single-element lookup table. This element is used to describe the packet which will be transmitted by the XPC-8 during an XID exchange. The TXID table is located immediately following the RLOOK table in system memory. Its format is identical to that of the TLOOK table element.

RECEIVER PASSWORD EXCHANGE LOOKUP TABLE

The RXID table is a single-element lookup table. This element is used to describe the buffer needed to store the packet received during and XID exchange. The RXID table is located immediately following the TXID table in system memory. Its format is identical to that of a RLOOK table element.

CHAPTER SEVEN

CONCLUSION

The development of this system provides us with the complete X.25 level-2 LAPB protocol. The design is proven by running test verification programs on the HP 4255A protocol analyzer. The next phase of this program would be to expand and put the next layer of X.25 on top of layer 2. This would involve setting up an X.25 level-3 protocol. This project involved work on the hardware and software level, hardware design and software design. The integration of the two elevated my knowledge of Computer Science by forcing me to consider the hardware problems associated with design. Usually one likes to program without consideration for the hardware which is the way it should be, but there are important issues involved in computer design which need the understanding of the system as a whole. And until hardware is capable of generic layout for the programmer an understanding of hardware issues is a must for all systems programmers.

VITA

Michael R. Kost Jr. was born in Bethlehem, Pennsylvania, on March 8, 1956. He is the son of Michael R. Kost and Agnes Marie Kost.

From 1969 through 1974, Mr. Kost attended Bethlehem Catholic High School in Bethlehem Pennsylvania. He graduated from East Stroudsburg University in East Stroudsburg Pennsylvania in 1984 with a B.S. in Computer Science.

He was a graduate student at Lehigh University from 1984 through 1987 in the Computer Science/Electrical Engineering department.

Mr. Kost works as a programmer at AT&T Technology Systems, Allentown, Pennsylvania. He is married to Deborah R. Kost and they have two children, Joshua and Sarah.

REFERENCES

1. CCITT - The International Telegraph and Telephone Consultative Committee. Red Book Volume VIII - Fascicle VIII.4 - Recommendation X.25
2. AT&T T7100 X.25 Protocol Controller Data Sheet
3. AT&T T7102 X.25/75 Protocol Controller Data Sheet

OPEN SYSTEM INTERFACE (OSI) MODEL

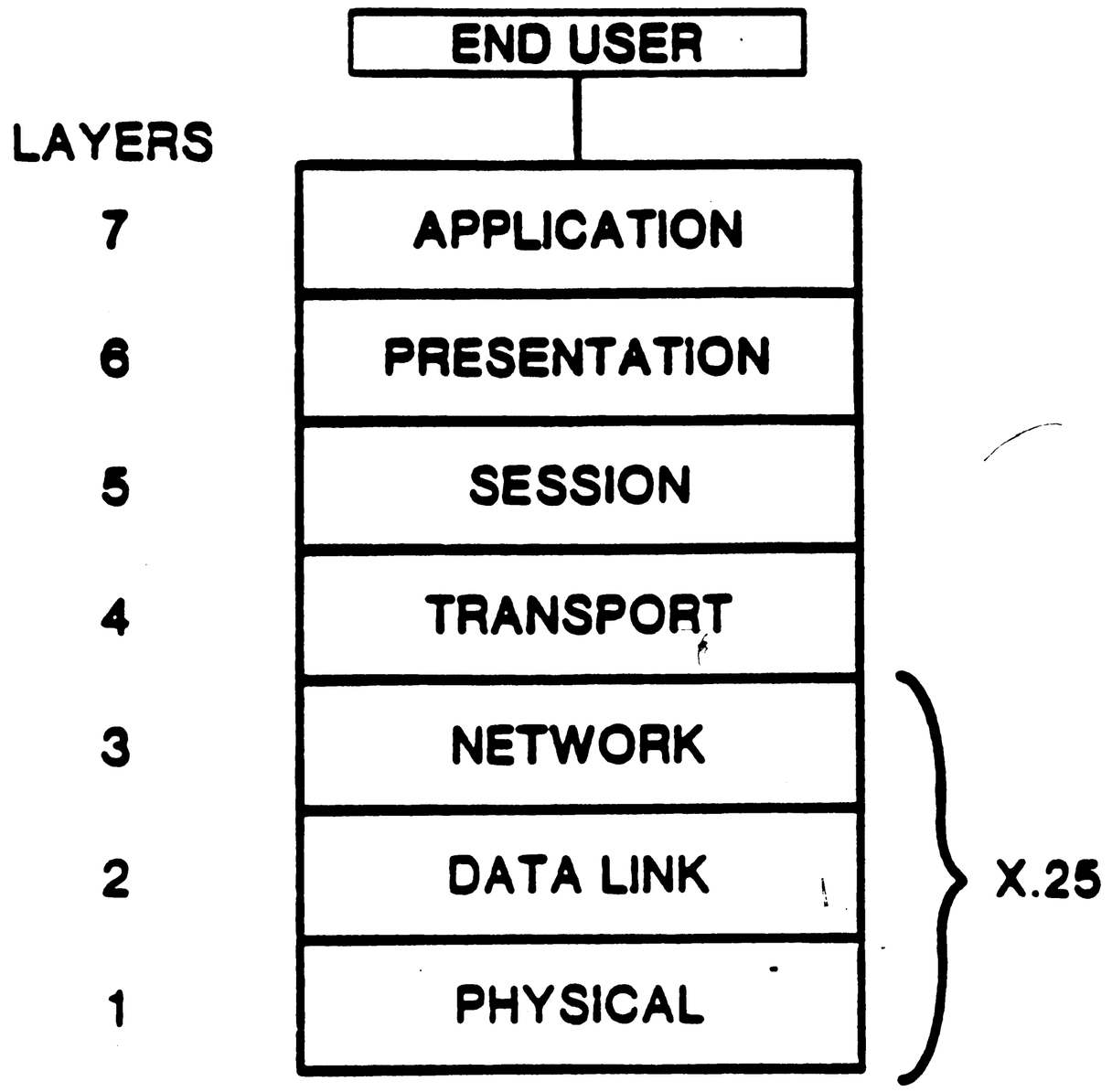


FIGURE 1

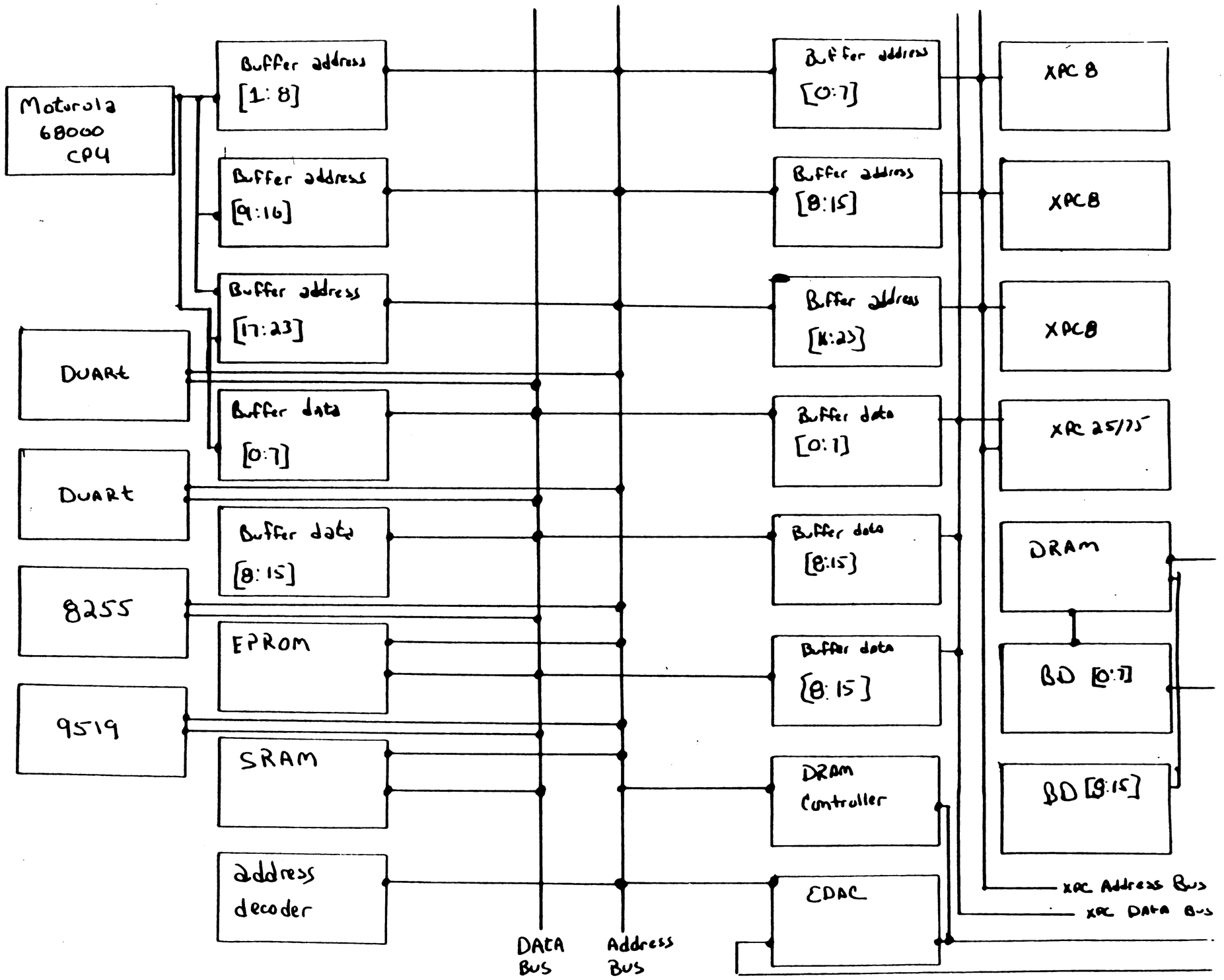
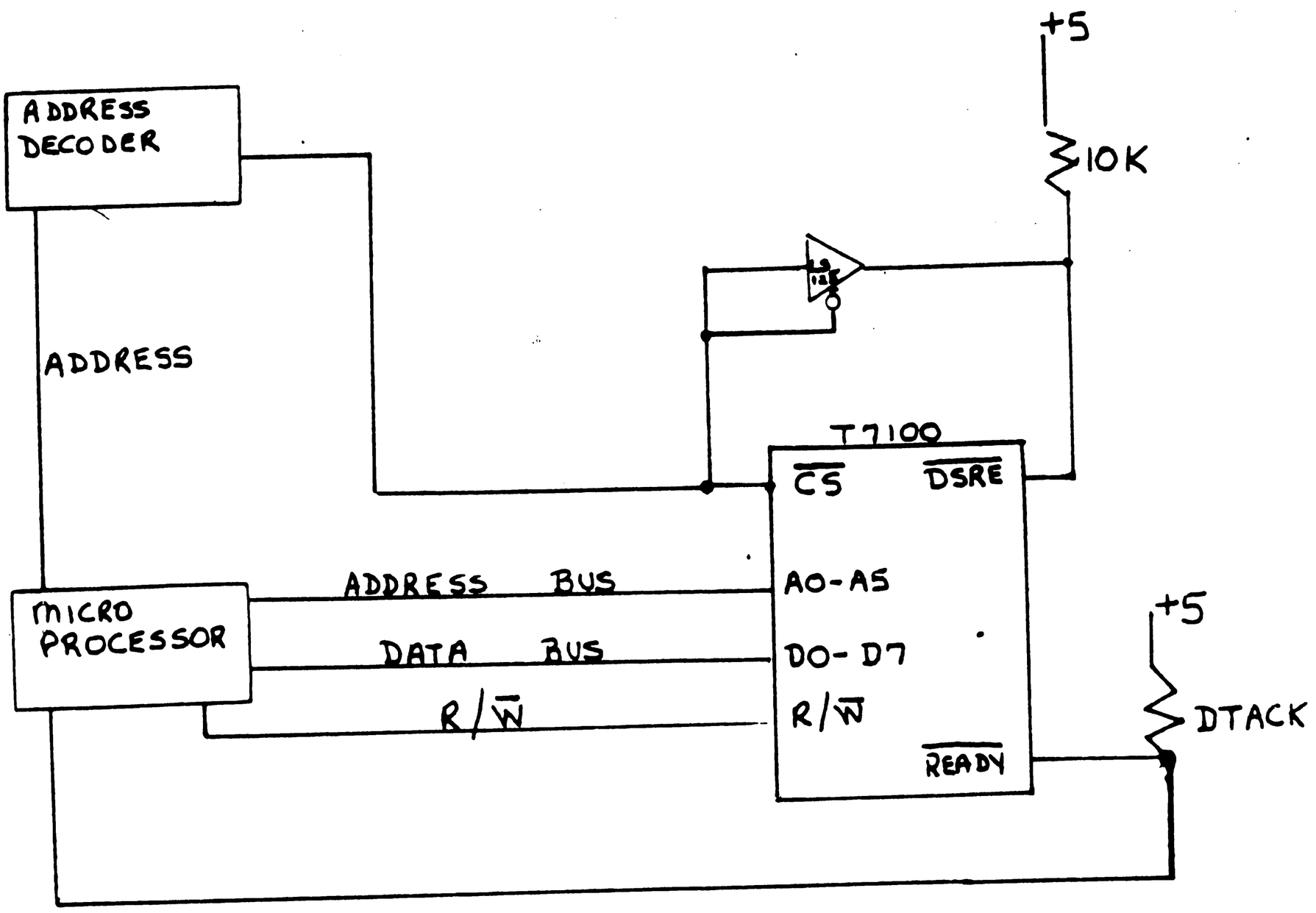

Motorola 68000 CPU
DUART
DUART
8255
9519
Buffer address [1:8]
Buffer address [9:16]
Buffer address [17:23]
Buffer data [0:7]
Buffer data [8:15]
EPROM
SRAM
address decoder
DATA Bus
Address Bus
Buffer address [0:7]
Buffer address [8:15]
Buffer address [16:23]
Buffer data [0:1]
Buffer data [8:15]
Buffer data [8:15]
DRAM Controller
EDAC
XAC 8
XAC 8
XAC 8
XAC 25/75
DRAM
BD [0:7]
BD [8:15]
XAC Address Bus
XAC Data Bus

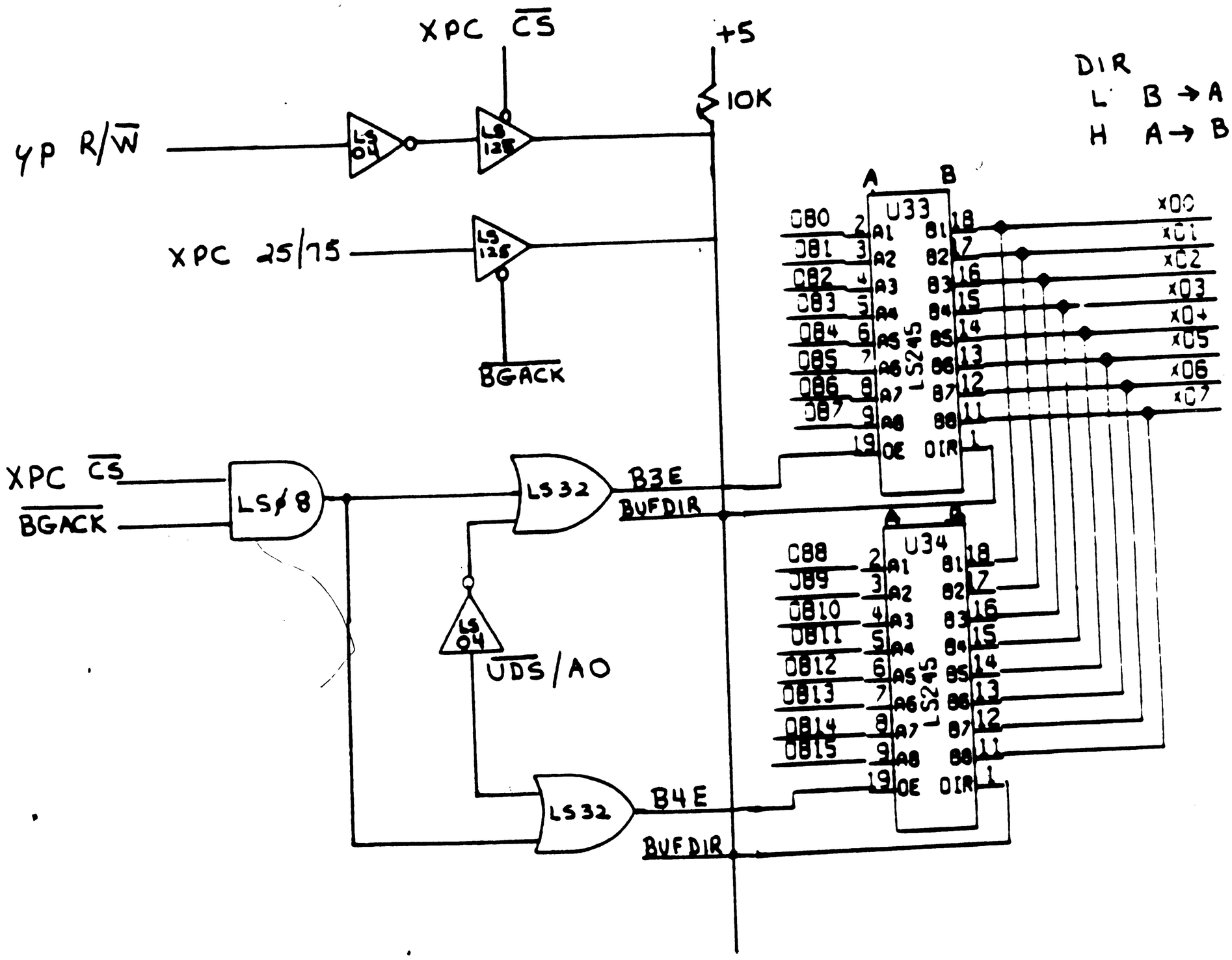
FIGURE 2

71.



REGISTER READ AND WRITE CIRCUITRY

FIGURE 3



DATA BUS INTERFACE CIRCUITRY

FIGURE 4

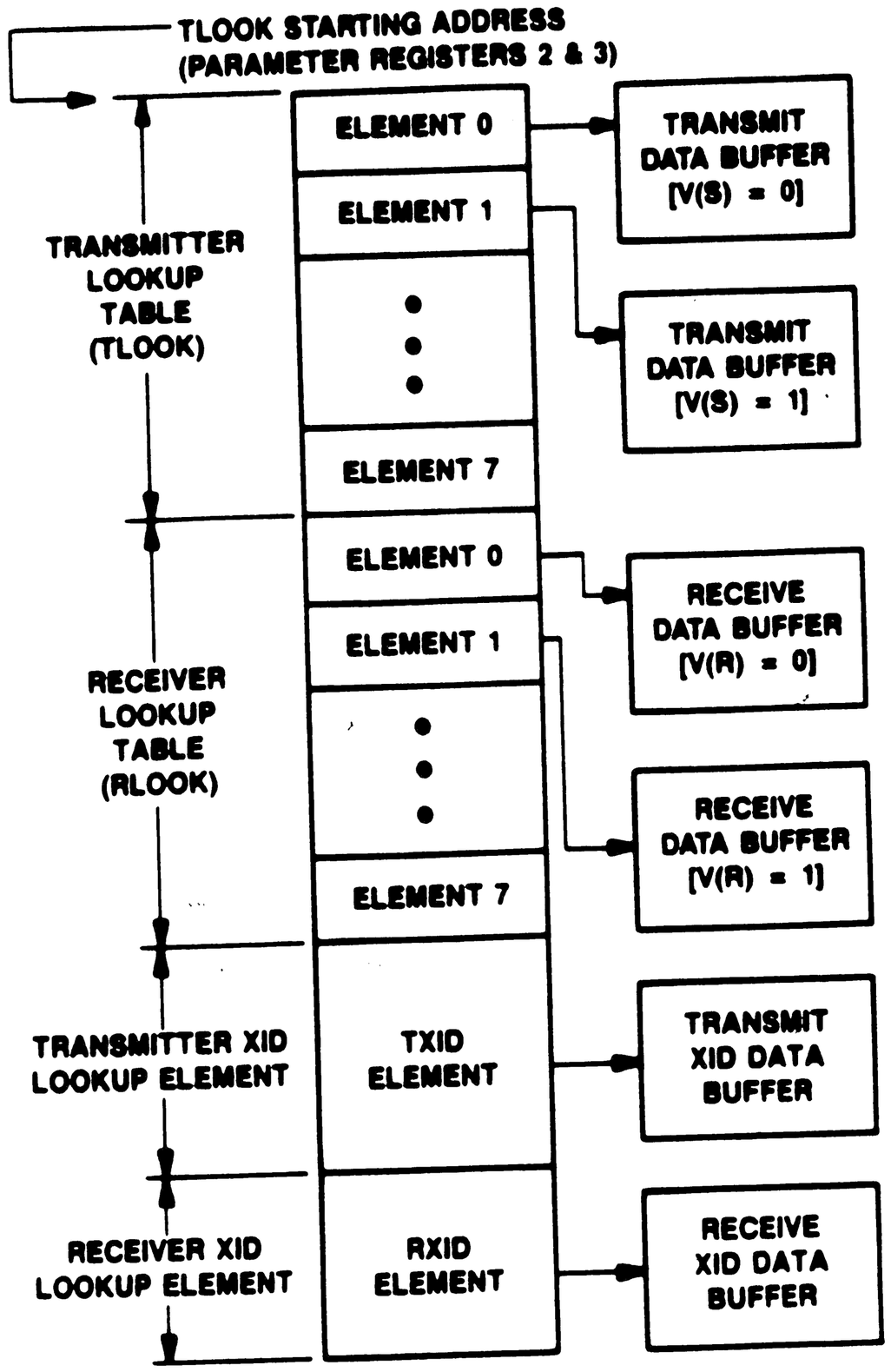


Figure 5. XPC-8/CPU Interface to System Memory

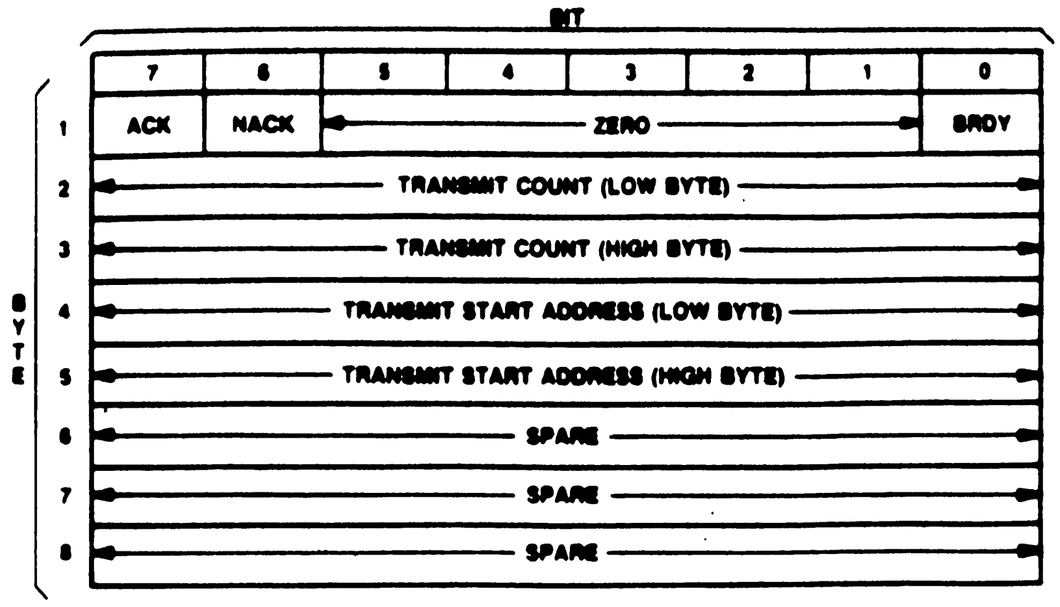


Figure 6. TLOOK ELEMENT

Table 1. TLOOK Element Definitions

Reg(s)	Bit(s)	Symbol	Name/Description
1	0	BRDY	**Buffer Ready.** Setting this bit to one (1) indicates to the XPC-8 that data associated with this element is ready to be transmitted. BRDY should be the last bit of the TLOOK element to be set by the CPU after preparing a buffer. The SEND bit of the Command Register should then be set to one (1) to command the XPC-8 to begin transmitting I frames. The XPC-8 clears BRDY and sets NACK after all of the data associated with this element has been accessed by DMA and loaded into the transmitter FIFO.
1	1–5	ZERO	**Zero.** These bits should be cleared to zero.
1	6	NACK	**Not Acknowledged.** The XPC-8 sets NACK and clears BRDY after all the data associated with this lookup element has been accessed and loaded into the transmitter FIFO. The XPC-8 clears this bit when the frame associated with this element has been acknowledged.
1	7	ACK	**Acknowledged.** The XPC-8 sets ACK and clears NACK when an acknowledgment is received for the packet associated with this lookup element. An XBA interrupt is generated to notify the CPU of one or more acknowledgments.
2	0–7	TCNT (Low Byte)	**Transmit Count.** The number of bytes in the transmit data field is specified by this 16-bit number. X.25 level 3 limits the size of the data field in a DATA packet to be not greater than 4K bytes. L3 adds a 3 or 4 byte header. The maximum size of the data field in an L2 I frame when X.25 L3 is used is 4K+4 bytes. In some applications L3 is not used. In any event the data field for the XPC-8 cannot exceed 8K bytes.
3	0–7	TCNT (High Byte)	
4	0–7	TSA (Low Byte)	**Transmit Start Address.** This 16-bit number is the location in system memory of the first byte of transmit data associated with this TLOOK element.
5	0–7	TSA (High Byte)	
6–8	0–7	SPARE	**Spare.** Not Used.

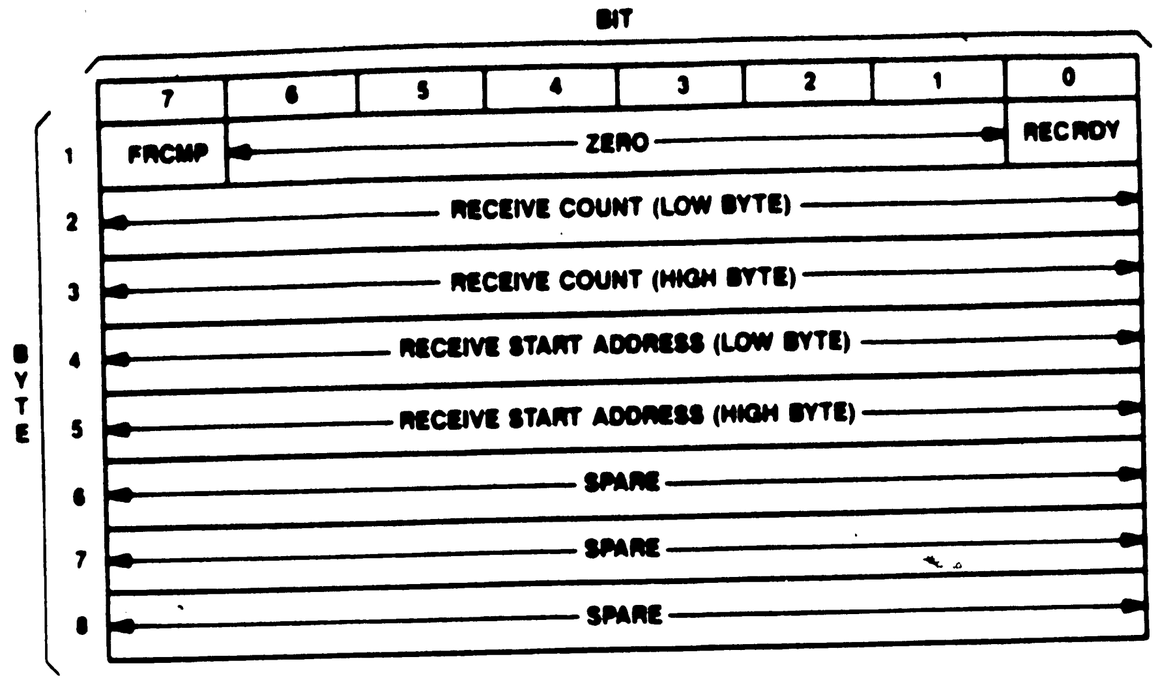


Figure 7. RLOOK Element

Table 2. RLOOK Element Definitions

Reg(s)	Bit(s)	Symbol	Name/Description
1	0	RECRDY	**Receiver Ready.** Setting this bit to one (1) indicates to the XPC-8 that the data buffer associated with this element is ready to receive data. The receive start address field of the RLOOK element should be specified before the CPU sets the RECRDY bit. After the XPC-8 receives a valid packet and stores the information field in the buffer associated with the lookup element, it clears RECRDY and sets FRCMP.
1	1–6	ZERO	**Zero.** These bits should be cleared to zero.
1	7	FRCMP	**Frame Complete.** When a valid packet is received, the XPC-8 writes the receive count, clears RECRDY, and sets FRCMP. A PKR interrupt is used to notify the CPU of the received packet.
2	0–7	RCNT (Low Byte)	**Receive Count.** This 16-bit number specifies how many bytes of system memory have been filled by the received packet. The XPC-8 writes this location after a valid I frame has been received and written to system memory without error (see counter N1).
3	0–7	RCNT (High Byte)	
4	0–7	RSA (Low Byte)	**Receive Start Address.** This 16-bit number specifies the address in system memory of the first byte of received data in the packet.
5	0–7	RSA (High Byte)	
6–8	0–7	SPARE	**Spare.** Not Used.

APPENDIX A

Appendix A contains the "C" source code used to initialize the communication board.

Appendix A also has the source code for the 68000 monitor. This source code has all the adjustments to run the communication board.

```
/*
        ************************************************************
        *                                                          *
        *       TUTOR BOARD - MRK 8/20/86                          *
        *       9519 INTERRUPT CONTROLLER                          *
        *       I/O DEFINITIONS                                    *
        *                                                          *
        ************************************************************
*/

/* point to 9519 at address 0x214000 */

#define USR_9519        ((struct intcntl *)0x214000)

        /*      AMD 9519 Interrupt Controller   */

/*
 *                                          7       0
 *                                         +---------+
 *      (struct intcntl *) -> data         |         |
 *                                         +---------+
 *
 *                                          7       0
 *                                         +---------+
 *                         -> command      |         |
 *                                         +---------+
 */

struct  intcntl
        {
        OFFSET;         unsigned char data;
        OFFSET;         unsigned char command;
        };


        /*  AMD 9519 command defines */

#define AMRESET         0x00
#define CLRALL          0x10
#define CLRIMRIRR       0x18
#define CLRIMR          0x20
#define CLRIMRX         0x28
#define SETIMR          0x30
#define SETIMRX         0x38
#define CLRIRR          0x40
#define CLRIRRX         0x48
#define SETIRR          0x50
#define SETIRRX         0x58
#define CLRHIISR        0x60
#define CLRISR          0x70
#define CLRISRX         0x71
#define LDMD40          0x80
#define LDMD56          0xa0
#define LDMD56S         0xa1
#define LDMD56C         0xa2
#define RDIMR           0xa4
#define LDIMR           0xb0
#define LDAUTOCLR       0xc0
#define LDMEM           0xe0


#define COUNT1          0x00    /* Byte count encoding */
#define COUNT2          0x08
#define COUNT3          0x10
#define COUNT4          0x18
```

```
#define VECT            0x07    /* Status Register bits */
#define ARMED           0x08
#define POLLED          0x10
#define ROTATING        0x20
#define ENABLED         0x40
#define IRRSET          0x80

#define ROTMODE         0x01    /* Mode register bits */
#define COMMVECT        0x02
#define POLLMODE        0x40
#define GINTHIGH        0x80
#define IREQHIGH        0x10
#define IRCSREG         0x00
#define IMRREG          0x20
#define IRRREG          0x40
#define ACRREG          0x60
#define MMARMED         0x80
```

```
/*****************************************************/
/*                                                   */
/*      TUTOR BOARD - MRK 9/9/86                     */
/*      8255 PERIPHERAL INTERFACE CONTROLLER         */
/*      I/O DEFINITIONS                              */
/*                                                   */
/*                                                   */
/*****************************************************/

/*  INTEL 8255 Peripheral Interface Controller */


#define PIC_8255 ((struct pic *)0x216000)

/*
 * Structure of an 8255:
 *                                                7      0
 *      (struct pic *) -> porta                  +--------+
 *                                               |        |
 *                                               +--------+
 *                     -> portb                  +--------+
 *                                               |        |
 *                                               +--------+
 *                     -> portc                  +--------+
 *                                               |        |
 *                                               +--------+
 *                     -> control                +--------+
 *                                               |        |
 *                                               +--------+
 */

struct pic
        {
        OFFSET;         unsigned char porta;
        OFFSET;         unsigned char portb;
        OFFSET;         unsigned char portc;
        OFFSET;         unsigned char control;
        };

/*
 *   8255 control defines
 */

#define ACTIVE  0x80
#define MODE0   0x00
#define MODE1   0x20
#define MODE2   0x40

#define PORTA_O 0x00    /* Port I/O defines */
#define PORTA_I 0x10
#define PORTB_O 0x00
#define PORTB_I 0x02
#define PORTC_O 0x00
#define PORTC_I 0x09
```

```
/*
**  FILE ID:            $HOME/tutor/68k/lib/dump_sta.c
**
**  Description:        Contians routines for printing out the
**                      status registers of the XPC-8's.
**
**  Functions:          dump_sta()
**
**  Memory:             Reads the xpc-8 status registers and
**                      writes the value to the screen.
**
**  Portability:        Expects init_structs() to set up the address
**                      dependences for xpc8[]. These are the physical
**                      addresses of the xpc-8's. Thus xpc8[] is an array
**                      of pointers to the xpc-8 devices.
**
**
*/


#include <globs.h>

void dump_sta()
{
int i;
        for ( i = 0; i <= 7; i++ )
        {
                printf("xpc8[1]->sta[%d] = %x ",i, xpc8[1]->sta[i]);
                printf("xpc8[2]->sta[%d] = %x ",i, xpc8[2]->sta[i]);
                printf("xpc8[3]->sta[%d] = %x \r\n",i, xpc8[3]->sta[i]);
        }
}
```

```
/* memory mapped address of XPC25/75's */
#define XPC2575 ((XPC25_75 *) 0x20e000)
#define TABLE_ADDR 0xc000
#define TABLE_2575 ((TABLE2575 *) 0xc000)

/* T7100 */

/* memory mapped address of XPC-8's */
#define XPC1 ((XPC_8 *) 0x206000)
#define XPC2 ((XPC_8 *) 0x208000)
#define XPC3 ((XPC_8 *) 0x20a000)

/* TLOOK starting address for XPC3 */
#define TABLE1_ADDR 0x8000
#define TABLE2_ADDR 0x8200
#define TABLE3_ADDR 0x8400
#define TABLE1 ((TABLE *) 0x8000)
#define TABLE2 ((TABLE *) 0x8200)
#define TABLE3 ((TABLE *) 0x8400)

/* create structure for XPC-8 registers */
typedef struct  {
        char    cmd;            /* XPC-8 command register */
        char    sta[8];         /* XPC-8 status registers */
        char    intp;           /* XPC-8 interrupt register */
        char    parm[17];       /* XPC-8 parameter registers */
        } XPC_8 ;


/* define the structure TLOOK as the template for Tlook elements */

typedef struct {                        /* REGISTER : NAME */
        /* NOTE:REVERSE ORDER OF 1ST BYTE */
        unsigned char ack    :1;        /* register 1 bit 7 : acknowledged */
        unsigned char nack   :1;        /* register 1 bit 6: not acknowledged
        unsigned char unused :5;        /* register 1 bits 1-5 : not used */
        unsigned char brdy   :1;        /* register 1 bit 0 : buffer ready */

        unsigned char tcntlow;  /* register 2 : Transmit Count low byte */
        unsigned char tcnthigh; /* register 3 : Transmit Count high byte */
        unsigned char tsalow;   /* register 4 : Transmit start address low by
        unsigned char tsahigh;  /* register 5 : Transmit start address high b
        unsigned char spare1;   /* unused */
        unsigned char spare2;   /* unused */
        unsigned char spare3;   /* unused */
        } TLOOK ;

typedef struct {                        /* REGISTER : NAME */
        /* NOTE:REVERSE ORDER OF BYTES to ASSIGN THE CORRECT BYTE ORDER */
        unsigned char frcmp   :1;       /* register 1 bit 7 : frame complete
        unsigned char unused  :6;       /* register 1 bits 1-6 : not used */
        unsigned char recrdy  :1;       /* register 1 bit 0 : reciever ready

        unsigned char rcntlow;  /* register 2 : Reciever Count low byte */
        unsigned char rcnthigh; /* register 3 : Reciever Count high byte */
        unsigned char rsalow;   /* register 4 : Receiver start address low by
        unsigned char rsahigh;  /* register 5 : Receiver start address high b
        unsigned char spare1;   /* unused */
        unsigned char spare2;   /* unused */
        unsigned char spare3;   /* unused */
        } RLOOK  ;

/* define the TLOOK Table which houses the 8 Tlook elements and 8 Rlook eleme

typedef struct{
```

```
        /* define the 8 element that make up the Transmitter Lookup Table */

         TLOOK tlook_element[8];

        /* define the 8 element that make up the Reciever Lookup Table */

        RLOOK rlook_element[8];

        /* define transmitter TXID lookup element */
        /* this is the same format as the TLOOK element */

        TLOOK txid;

        /* define transmitter RXID lookup element */
        /* this is the same format as the RLOOK element */

        RLOOK rxid;

        }  TABLE ;

/*****************************************************************************/
/* T7102 */

/* create structure for XPC25/75  registers */
typedef struct  {
        char    cmd0;           /* XPC25/75 command register */
        char    cmd;            /* XPC25/75 command register */
        char    sta[28];        /* XPC25/75 status registers */
        char    intp0;          /* XPC25/75 interrupt register */
        char    intp;           /* XPC25/75 interrupt register */
        char    parm[32];       /* XPC25/75 parameter registers */
        char    cntr[36];       /* XPC25/75 counter registers */
        char    parm016;        /* XPC25/75 parameter registers */
        char    parm16;         /* XPC25/75 parameter registers */
        } XPC25_75 ;


/* define the structure TLOOK as the template for Tlook elements */
/* ASSUME A SIZE TLOOK & RLOOK TABLE OF 8 (N=9) */

typedef struct {                        /* REGISTER : NAME */
        /* NOTE:REVERSE ORDER OF 1ST BYTE */
        unsigned char ack     :1;       /* register 1 bit 7 : acknowledged */
        unsigned char nack    :1;       /* register 1 bit 6: not acknowledged
        unsigned char unused  :2;       /* register 1 bits 4-5 : not used */
        unsigned char resd    :3;       /* register 1 bit 1-3 : residual */
        unsigned char brdy    :1;       /* register 1 bit 0 : buffer ready */

        unsigned char tcntlow;          /* register 2 : Transmit Count low by
        unsigned char spare0  :4;       /* register 3 : bits 4-7 */
        unsigned char tcnthigh :4;      /* register 3 : bits 0-4 Transmit Cou
        unsigned char tsalow;   /* register 4 : Transmit start address low by
        unsigned char tsamid;   /* register 5 : Transmit start address middle
        unsigned char tsahigh;  /* register 6 : Transmit start address high b
        unsigned char spare1;   /* unused */
        unsigned char spare2;   /* unused */
        } TLOOK2575 ;

typedef struct {                        /* REGISTER : NAME */
        /* NOTE:REVERSE ORDER OF BYTES to ASSIGN THE CORRECT BYTE ORDER */
        unsigned char frcmp    :1;      /* register 1 bit 7 : frame complete
        unsigned char unused   :1;      /* register 1 bit 6 : not used */
        unsigned char badid    :1;      /* register 1 bit 5 : bad id field */
        unsigned char unused1  :1;      /* register 1 bit 4 : not used */
        unsigned char resd     :3;      /* register 1 bit 1-3 : residual */
```

```
unsigned char recrdy  :1;       /* register 1 bit 0 : reciever ready

unsigned char rcntlow;  /* register 2 : Reciever Count low byte */
unsigned char spare0 :4;        /* register 3 : bits 4-7 */
unsigned char rcnthigh :4;      /* register 3 : bits 0-4 Transmit Cou
unsigned char rsalow;   /* register 4 : Receiver start address low by
unsigned char rsamid;   /* register 5 : Transmit start address middle
unsigned char rsahigh;  /* register 5 : Receiver start address high b
unsigned char spare1;   /* unused */
unsigned char spare2;   /* unused */
} RLOOK2575  ;

/* define the TLOOK Table which houses the 8 Tlook elements and 8 Rlook eleme

typedef struct{

        /* define the 8 element that make up the Transmitter Lookup Table */

         TLOOK2575 tlook_2575[8];

        /* define the 8 element that make up the Reciever Lookup Table */

        RLOOK2575 rlook_2575[8];

        /* define transmitter TXID lookup element */
        /* this is the same format as the TLOOK element */

        TLOOK2575 txid;

        /* define transmitter RXID lookup element */
        /* this is the same format as the RLOOK element */

        RLOOK2575 rxid;

        }  TABLE2575 ;
```

```
#define TRUE    1
#define FALSE   0

#define OFFSET  unsigned :8

#define CR      "\r\n"
#define NL      "\n"
#define RETURN  "\r"

#define ENABLE 0x01
#define DISABLE 0x00

#define xpc1 0
#define xpc2 1
#define xpc3 2

#define tlook1 0
#define tlook2 1
#define tlook3 2
```

```
#include <stdio68.h>
#include <tutor.h>
#include <xpc8.h>
#include <typedefs.h>

extern XPC_8 *xpc8[];
extern TABLE *tables[];

/* Buffer length 0x100 hex (256 decimal) */
/* defined by N1 parameter register 8-9(bits 0-4)  */
#define BUFFER_size 0X100

/* start of XPC8's buffer address`s */
#define BUFFER_ADDR 0x8600

/* start of XPC2575's buffer address`s */
#define BUFF_ADDR2575 0xc600
```

```
/*
**  FILE ID:            $HOME/tutor/68k/lib/init_dev.c
**
**  Description:        Contians routines for initializing the following
**                      devices:
**                              8255
**                              upper address bit register for xpc-8's
**
**  Functions:          init_8255()
**                      init_up_add_bit()
**
**  Memory:             Expects the 8255 to be at the memory location
**                      defined in 8255.h. Also the device for upper
**                      address bits to be in specific areas of memory.
**
**
**  Portability:        Expects specific address defined here and in 8255.h.
**
**
*/


#include <globs.h>
#include <8255.h>

/*****************************************************************************
**
**  Function:           init_8255()
**
**  Description:        Initialize port a,b,c of the 8255 for baud rate.
**
**  Inputs:             None.
**
**  Returns:            None.
**
**  Calls:              None.
**
**  Portability:        PIC_8255-> is a pointer to the memory location of
**                      the 8255 on the tutor board. This is defined in
**                      8255.h.
**
*/

init_8255()
{
        PIC_8255->control = 0x80;
        PIC_8255->porta = 0x1b;
        PIC_8255->portb = 0x1b;
        PIC_8255->portc = 0xff;
}

/*****************************************************************************
**
**  Function:           init_up_add_bit()
**
**  Description:        Initialize the upper addresses of the xpc8 devices.
**
**  Inputs:             None.
**
**  Returns:            None.
**
**  Calls:              None.
**
**  Portability:        Expects the devices to be at specific address.
**
```

```
*/

init_up_add_bit()
{

        char *reg1, *reg2, *reg3;

        reg1 = ((char *) 0x21a000);
        reg2 = ((char *) 0x21c000);
        reg3 = ((char *) 0x21e000);
        *reg1 = 0x00;
        *reg2 = 0x00;
        *reg3 = 0x00;

        reg1 = ((char *) 0x21a001);
        reg2 = ((char *) 0x21c001);
        reg3 = ((char *) 0x21e001);
        *reg1 = 0x00;
        *reg2 = 0x00;
        *reg3 = 0x00;
}
```

```
/*
**  FILE ID:            $HOME/tutor/68k/lib/init_xpc8.c
**
**  Description:        Initialization routines for the xpc8 devices,
**                      and the xpc parameter registers. Build TLOOK tables,
**                      and set the tlook brdy bit. Set up some transmit pack
**
**
**  Functions:          init_structs()
**                      initialize_xpc8()
**                      init_table()
**                      set_brdys()
**                      init_cmd()
**                      txmpkts()
**                      strcpy()
**
**  Memory:
**
**
**  Portability:        Expects the devices are set up at specific addresses
**                      defined in typedefs.h.
**
**
**  Last Modification:  mrk 12/4/86
*/

#include <globs.h>

/*****************************************************************************
**
**  Function:           init_structs()
**
**  Description:        Build array of pointers to the xpc8 devices.
**
**  Inputs:             None.
**
**  Returns:            None.
**
**  Calls:              None.
**
**  Portability:        XPC1, XPC2, XPC3 are define to specific areas of memo
**                      defined in typedefs.h.
**
**
*/

init_structs()
{
        xpc8[xpc1] = XPC1;
        xpc8[xpc2] = XPC2;
        xpc8[xpc3] = XPC3;

        tables[tlook1] = TABLE1;
        tables[tlook2] = TABLE2;
        tables[tlook3] = TABLE3;

}

/*****************************************************************************
**
**  Function:           initialize_xpc8()
**
**  Description:        Initialize parameter registers of the xpc8 devices.
**
**  Inputs:             None.
```

```
**
**  Returns:            None.
**
**  Calls:              None.
**
**  Portability:        Xpc8[] is define to specific areas of memory,
**                      defined in init_structs().
**
**
*/
initialize_xpc8 ()

{

int i, xpc;

        /* initialize the parameter registers */

/* xpc1 */

        xpc8[xpc1]->parm[0] = 0x00;     /* PR0  - Loopback and echo bits off
        xpc8[xpc1]->parm[1] = 0x04;     /* PR1  - Window size = 4, XID mode =
                                        /* PR2  - TOOK start address low byte
        xpc8[xpc1]->parm[2] = TABLE1_ADDR & 0xff;
                                        /* PR3  - TOOK start address high byt
        xpc8[xpc1]->parm[3] = TABLE1_ADDR >> 8;
        xpc8[xpc1]->parm[4] = 0xA9;     /* PR4  - Acknowledgement timer T1 co
                                           count low byte; here T1 = 3sec??
        xpc8[xpc1]->parm[5] = 0x30;     /* PR5  - N2 = 3; T1 count high byte
        xpc8[xpc1]->parm[6] = 0x9E;     /* PR6  - Respsonse delay timer T2 co
                                           byte;  T2 here = 2.5 seconds
        xpc8[xpc1]->parm[7] = 0x00;     /* PR7  - T2 timer count high byte
        xpc8[xpc1]->parm[8] = 0x00;     /* PR8  - N1 count low byte
        xpc8[xpc1]->parm[9] = 0x01;     /* PR9  - N1 count high byte
        xpc8[xpc1]->parm[10] = 0x65;    /* PR10 - Inactive link timer T4
                                           low byte here; T4 =10 sec.
        xpc8[xpc1]->parm[11] = 0x04;    /* PR11 - T4 timer high byte
        xpc8[xpc1]->parm[12] = 0x7F;    /* PR12 - Idle link timer T3 count lo
                                           byte; T3 = 25 sec.
        xpc8[xpc1]->parm[13] = 0x05;    /* PR13 - T3 timer count high byte
        xpc8[xpc1]->parm[14] = 0x01;    /* PR14 - Link Transmit command addre
        xpc8[xpc1]->parm[15] = 0x03;    /* PR15 - Link Transmit response addr
        xpc8[xpc1]->parm[16] = 0x00;    /* PR16 - Extra flag count

/* xpc2 */

        xpc8[xpc2]->parm[0] = 0x00;     /* PR0  - Loopback and echo bits off
        xpc8[xpc2]->parm[1] = 0x04;     /* PR1  - Window size = 4, XID mode =
                                        /* PR2  - TOOK start address low byte
        xpc8[xpc2]->parm[2] = TABLE2_ADDR & 0xff;
                                        /* PR3  - TOOK start address high byt
        xpc8[xpc2]->parm[3] = TABLE2_ADDR >> 8;
        xpc8[xpc2]->parm[4] = 0xA9;     /* PR4  - Acknowledgement timer T1 co
                                           count low byte; here T1 = 3sec??
        xpc8[xpc2]->parm[5] = 0x30;     /* PR5  - N2 = 3; T1 count high byte
        xpc8[xpc2]->parm[6] = 0x9E;     /* PR6  - Respsonse delay timer T2 co
                                           byte;  T2 here = 2.5 seconds
        xpc8[xpc2]->parm[7] = 0x00;     /* PR7  - T2 timer count high byte
        xpc8[xpc2]->parm[8] = 0x00;     /* PR8  - N1 count low byte
        xpc8[xpc2]->parm[9] = 0x01;     /* PR9  - N1 count high byte
        xpc8[xpc2]->parm[10] = 0x65;    /* PR10 - Inactive link timer T4
                                           low byte here; T4 =10 sec.
        xpc8[xpc2]->parm[11] = 0x04;    /* PR11 - T4 timer high byte
        xpc8[xpc2]->parm[12] = 0x7F;    /* PR12 - Idle link timer T3 count lo
                                           byte; T3 = 25 sec.
        xpc8[xpc2]->parm[13] = 0x05;    /* PR13 - T3 timer count high byte
```

```
        xpc8[xpc2]->parm[14] = 0x01;    /* PR14 - Link Transmit command addre
        xpc8[xpc2]->parm[15] = 0x03;    /* PR15 - Link Transmit response addr
        xpc8[xpc2]->parm[16] = 0x00;    /* PR16 - Extra flag count

/* xpc3 */


        xpc8[xpc3]->parm[0] = 0x00;     /* PR0  - Loopback and echo bits off
        xpc8[xpc3]->parm[1] = 0x04;     /* PR1  - Window size = 4, XID mode =
                                /* PR2  - TOOK start address low byte     */
        xpc8[xpc3]->parm[2] = TABLE3_ADDR & 0xff;
                                        /* PR3  - TOOK start address high byt
        xpc8[xpc3]->parm[3] = TABLE3_ADDR >> 8;
        xpc8[xpc3]->parm[4] = 0xA9;     /* PR4  - Acknowledgement timer T1 co
                                           count low byte; here T1 = 3sec??
        xpc8[xpc3]->parm[5] = 0x30;     /* PR5  - N2 = 3; T1 count high byte =
        xpc8[xpc3]->parm[6] = 0x9E;     /* PR6  - Respsonse delay timer T2 co
                                           byte;  T2 here = 2.5 seconds
        xpc8[xpc3]->parm[7] = 0x00;     /* PR7  - T2 timer count high byte
        xpc8[xpc3]->parm[8] = 0x00;     /* PR8  - N1 count low byte
        xpc8[xpc3]->parm[9] = 0x01;     /* PR9  - N1 count high byte
        xpc8[xpc3]->parm[10] = 0x65;    /* PR10 - Inactive link timer T4
                                           low byte here; T4 =10 sec.
        xpc8[xpc3]->parm[11] = 0x04;    /* PR11 - T4 timer high byte
        xpc8[xpc3]->parm[12] = 0x7F;    /* PR12 - Idle link timer T3 count lo
                                           byte; T3 = 25 sec.
        xpc8[xpc3]->parm[13] = 0x05;    /* PR13 - T3 timer count high byte
        xpc8[xpc3]->parm[14] = 0x03;    /* PR14 - Link Transmit command addre
        xpc8[xpc3]->parm[15] = 0x01;    /* PR15 - Link Transmit response addr
        xpc8[xpc3]->parm[16] = 0x00;    /* PR16 - Extra flag count

}

/*****************************************************************************
**
**  Function:           init_table()
**
**  Description:        Build the TLOOK tables. Define the Tlook elements and
**                      the pointers to the data buffers. Also set up TXID/RX
**
**  Inputs:             None.
**
**  Returns:            None.
**
**  Calls:              None.
**
**  Portability:        Builds the data buffers starting at BUFFER_ADDR
**                      (defined in globs.h). Disables and clears the
**                      corresponding element bits.
**
**
*/
init_table()

{
        /* assume 16 bit unsigned integer holds buffer address */
        unsigned int buff_addr;

        int i, table;


        buff_addr = BUFFER_ADDR;
        for ( table = 0; table < 3; table++ )
        {
```

```
for ( i = 0; i <= 7; i++ )
{
        /* Transmit Buffer Table */
        /* disable and clear the corresponding TLOOK element
        tables[table]->tlook_element[i].brdy = DISABLE;
        tables[table]->tlook_element[i].nack = DISABLE;
        tables[table]->tlook_element[i].ack = DISABLE;
        tables[table]->tlook_element[i].tcntlow = 0x26;
        tables[table]->tlook_element[i].tcnthigh = 0x00;
        tables[table]->tlook_element[i].tsalow = buff_addr;
        tables[table]->tlook_element[i].tsahigh = buff_addr >
        buff_addr = buff_addr + BUFFER_size;
};

for ( i = 0; i <= 7; i++ )
{
        /* point to the ith Rlook element of TABLE */

        /* disable and clear the corresponding RLOOK element '

        tables[table]->rlook_element[i].frcmp = DISABLE;
        tables[table]->rlook_element[i].rcntlow = 0x00;
        tables[table]->rlook_element[i].rcnthigh = 0x00;
        tables[table]->rlook_element[i].rsalow = buff_addr;
        tables[table]->rlook_element[i].rsahigh = buff_addr >
        tables[table]->rlook_element[i].recrdy = ENABLE;
        buff_addr = buff_addr + BUFFER_size;
    };

    /* add buffer for TXID element */
    buff_addr = buff_addr + BUFFER_size;

    /* add buffer for RXID element */
    buff_addr = buff_addr + BUFFER_size;
  };
}

/*****************************************************************************
**
**  Function:           set_brdys()
**
**  Description:
**
**  Inputs:             None.
**
**  Returns:            None.
**
**  Calls:              None.
**
**  Portability:        XPC1, XPC2, XPC3 are define to specific areas of memor
**                      defined in typedefs.h.
**
*/
set_brdys()
{
        /* all xpc all brdys*/
        int table, i;

        for ( table = 0; table < 3; table++ )
        {
                for ( i = 0; i <= 7; i++ )
                {
                        tables[table]->tlook_element[i].brdy = 0x1;
```

```
        };
    };
}

/*****************************************************************************
**
**  Function:           init_cmd()
**
**  Description:
**
**  Inputs:             None.
**
**  Returns:            None.
**
**  Calls:              None.
**
**  Portability:        XPC1, XPC2, XPC3 are define to specific areas of memo
**                      defined in typedefs.h.
**
*/

init_cmd()
{
        xpc8[xpc1]->cmd = 0x8d;
        xpc8[xpc2]->cmd = 0x8d;
        xpc8[xpc3]->cmd = 0x8d;
        printf(" after 0x8d write to XPC1->cmd = %x\r\n",XPC1->cmd);
        printf(" after 0x8d write to XPC2->cmd = %x\r\n",XPC2->cmd);
        printf(" after 0x8d write to XPC3->cmd = %x\r\n",XPC3->cmd);
}
```

```
/*
**  FILE ID:            $HOME/tutor/68k/lib/txmpkts.c
**
**  Description:        Transmit packets are set up at addresses which
**                      are defined in the tlook elements (transmitter
**                      lookup table). Here is where we force the packets at
**                      the addresses defined in the function init_table().
**
**  Functions:          txmpkts()
**                      strcpy()
**
**  Memory:             The area defined in the tlook elements, the transmit
**                      data buffers N(s)=0 through N(s)=7.
**
*/

txmpkts()
{
unsigned char  buff;


        /*XPC3 transmit packets */
        strcpy( ((char *) 0x8600), "This is packet 0 from XPC1");
        strcpy( ((char *) 0x8700), "This is packet 1 from XPC1");
        strcpy( ((char *) 0x8800), "This is packet 2 from XPC1");
        strcpy( ((char *) 0x8900), "This is packet 3 from XPC1");
        strcpy( ((char *) 0x8a00), "This is packet 4 from XPC1");
        strcpy( ((char *) 0x8b00), "This is packet 5 from XPC1");
        strcpy( ((char *) 0x8c00), "This is packet 6 from XPC1");
        strcpy( ((char *) 0x8d00), "This is packet 7 from XPC1");

        /*XPC2 transmit packets */
        strcpy( ((char *) 0x9800), "This is a packet 0 from XPC2");
        strcpy( ((char *) 0x9900), "This is a packet 1 from XPC2");
        strcpy( ((char *) 0x9a00), "This is a packet 2 from XPC2");
        strcpy( ((char *) 0x9b00), "This is a packet 3 from XPC2");
        strcpy( ((char *) 0x9c00), "This is a packet 4 from XPC2");
        strcpy( ((char *) 0x9d00), "This is a packet 5 from XPC2");
        strcpy( ((char *) 0x9e00), "This is a packet 6 from XPC2");
        strcpy( ((char *) 0x9f00), "This is a packet 7 from XPC2");

        /*XPC3 transmit packets */
        strcpy( ((char *) 0xaa00), "This is a packet 0 from XPC3");
        strcpy( ((char *) 0xab00), "This is a packet 1 from XPC3");
        strcpy( ((char *) 0xac00), "This is a packet 2 from XPC3");
        strcpy( ((char *) 0xad00), "This is a packet 3 from XPC3");
        strcpy( ((char *) 0xae00), "This is a packet 4 from XPC3");
        strcpy( ((char *) 0xaf00), "This is a packet 5 from XPC3");
        strcpy( ((char *) 0xb000), "This is a packet 6 from XPC3");
        strcpy( ((char *) 0xb100), "This is a packet 7 from XPC3");


}
 strcpy(s, t) /* copy t to s */
 char *s, *t;
{
        while ( *s++ = *t++)
                ;
}
```

```
/*
**  FILE ID:            $HOME/tutor/68k/lib/init.c
**
**  Description:        Initialization routines for the T7102 device,
**                      and the parameter registers. Build TLOOK tables,
**                      and set the tlook brdy bit. Set up some transmit pack
**
**  Functions:          _initialize()
**                      _table()
**                      _brdys()
**                      _cmd()
**                      _txmpkts()
**
**  Memory:
**
**  Portability:        Expects the devices are set up at specific addresses
**                      defined in typedefs.h.
**
**  Last Modification:  mrk 1-27-87
*/

#include <globs.h>

/*****************************************************************************
**
**  Function:           do_()
**
**  Description:        Initialize T7102 device.
**
**  Inputs:             None.
**
**  Returns:            None.
**
**  Calls:              _initialize()
**                      _table()
**                      _brdys()
**                      _cmd()
**                      _txmpkts()
**
**  Portability:        Xpc8[] is define to specific areas of memory,
**                      defined in init_structs().
**
*/

go_2575()
{
        printf("\r\nINIT_2575\r\n\r\n");
        printf("SET XPC2575->cmd to 0x82\r\n");
        XPC2575->cmd = 0x82;
        _txmpkts();
        _initialize();
        _table();
        _brdys();
        _cmd();
        dump_2575_sta();

}
```

```
/*****************************************************************************
**
**  Function:           _initialize()
**
**  Description:        Initialize parameter registers of the T7102 device.
**
**  Inputs:             None.
**
**  Returns:            None.
**
**  Calls:              None.
**
**  Portability:        Xpc8[] is define to specific areas of memory,
**                      defined in init_structs().
**
*/
_initialize()
{

int i, xpc;

        /* initialize the parameter registers */

/* xpc1 */

        XPC2575->parm[0] = 0x00;        /* PR0  - Loopback and echo bits off
        XPC2575->parm[2] = 0x04;        /* PR1  - Modulus 8, parity disabled,
                                           flags when disc, no counter int, no
                                           16-bit CRC, X.25 mode */

                                        /* PR2  -  transmit window size = 7
        XPC2575->parm[4] = 0x03;

                                        /* PR3-5 - TOOK start address */
        XPC2575->parm[6] = TABLE_ADDR & 0xff;
        XPC2575->parm[8] = TABLE_ADDR >> 8;
        XPC2575->parm[10] = TABLE_ADDR >> 16;

        XPC2575->parm[12] = 0xa3;       /* PR6  - Respsonse delay timer T1 co
                                           byte;  */
        XPC2575->parm[14] = 0x32;       /* PR7  - T1 timer count high byte;
                                        retransmission counter */
        XPC2575->parm[16] = 0x98;       /* PR8  - T3 count low byte
        XPC2575->parm[18] = 0x1e;       /* PR9  - T3 count high byte; interfr
                                        flag fill*/
        XPC2575->parm[20] = 0x00;       /* PR10 - N1 -low byte */
        XPC2575->parm[22] = 0x08;       /* PR11 - N1 high byte */
        XPC2575->parm[24] = 0x01;       /* PR12 - Transmit command address */
        XPC2575->parm[26] = 0x03;       /* PR13 - Transmit response address *
        XPC2575->parm[28] = 0x42;       /* PR14 -  Byte order = Motorola
                                        DMA; Limit */
        XPC2575->parm[30] = 0x20;       /* PR15 - BPChar; Header; Byte mode
                                        UNFCTL */
        XPC2575->parm16 = 0x00; /* PR16 - Extra flag count                  */


}

/*****************************************************************************
**
**  Function:           init_table()
**
**  Description:        Build the TLOOK tables. Define the Tlook elements and
**                      the pointers to the data buffers. Also set up TXID/RX
**
```

```
**  Inputs:             None.
**
**  Returns:            None.
**
**  Calls:              None.
**
**  Portability:        Builds the data buffers starting at BUFFER_ADDR
**                      (defined in globs.h). Disables and clears the
**                      corresponding element bits.
**
**
*/

_table()
{
        /* assume 16 bit unsigned integer holds buffer address */
        unsigned int buff_addr;

        int i, table;


        buff_addr = BUFF_ADDR2575;

        for ( i = 0; i <= 7; i++ )
        {

                /* Transmit Buffer Table */
                /* disable and clear the corresponding TLOOK element */
                TABLE_2575->tlook_2575[i].brdy = 0;
                TABLE_2575->tlook_2575[i].resd = 0;
                TABLE_2575->tlook_2575[i].nack = DISABLE;
                TABLE_2575->tlook_2575[i].ack = DISABLE;
                TABLE_2575->tlook_2575[i].tcntlow = 0x26;
                TABLE_2575->tlook_2575[i].tcnthigh = 0x0;
                TABLE_2575->tlook_2575[i].tsalow = buff_addr;
                TABLE_2575->tlook_2575[i].tsamid = buff_addr >> 8;
                TABLE_2575->tlook_2575[i].tsahigh = buff_addr >> 16;
                buff_addr = buff_addr + BUFFER_size;
        };

        for ( i = 0; i <= 7; i++ )
        {
                /* point to the ith Rlook element of TABLE */

                /* disable and clear the corresponding RLOOK element */

                TABLE_2575->rlook_2575[i].frcmp = 0;
                TABLE_2575->rlook_2575[i].badid = 0;
                TABLE_2575->rlook_2575[i].rcntlow = 0x00;
                TABLE_2575->rlook_2575[i].rcnthigh = 0x0;
                TABLE_2575->rlook_2575[i].rsalow = buff_addr;
                TABLE_2575->rlook_2575[i].rsamid = buff_addr >> 8;
                TABLE_2575->rlook_2575[i].rsahigh = buff_addr >> 16;
                TABLE_2575->rlook_2575[i].resd = 0x0;
                TABLE_2575->rlook_2575[i].recrdy = ENABLE;
                buff_addr = buff_addr + BUFFER_size;
        };

        /* add buffer for TXID element */
        buff_addr = buff_addr + BUFFER_size;

        /* add buffer for RXID element */
        buff_addr = buff_addr + BUFFER_size;

}
```

```
/*****************************************************************************
**
**  Function:           _brdys()
**
**  Description:
**
**  Inputs:             None.
**
**  Returns:            None.
**
**  Calls:              None.
**
**  Portability:        XPC1, XPC2, XPC3 are define to specific areas of memo
**                      defined in typedefs.h.
**
**
*/
 _brdys()
{
        /* set all xpc2575 brdys */
        int i;

        for ( i = 0; i <= 7; i++ )
        {
                TABLE_2575->tlook_2575[i].brdy = 1;

        };
}

/*****************************************************************************
**
**  Function:           2575_cmd()
**
**  Description:        Build array of pointers to the xpc8 devices.
**
**  Inputs:             None.
**
**  Returns:            None.
**
**  Calls:              None.
**
**  Portability:        XPC_2575 is define at specific areas of memory,
**                      defined in typedefs.h.
**
**
*/
 _cmd()
{

        XPC2575->cmd = 0x8d;
        printf(" after 8d write to XPC2575->cmd = %x\r\n",XPC2575->cmd);

}

/*****************************************************************************

/*
**  Function:           dump_2575_sta()
**
**  Description:        Contians routines for printing out the
**                      status registers of the XPC-8's.
**
**
**  Memory:             Reads the xpc-8 status registers and
**                      writes the value to the screen.
**
**
**  Portability:        Expects init_structs() to set up the address
**                      dependences for xpc8[]. These are the physical
```

```
**                      addresses of the xpc-8's. Thus xpc8[] is an array
**                      of pointers to the xpc-8 devices.
**
**
**
*/


 dump_2575_sta()
{
int i;
        printf("\r\nPARAMETER REGISTERS\r\n");
        for ( i = 0; i <= 32; i = i + 2 )
        {
                printf("xpc2575->parm[%d] = %x\r\n ",i, XPC2575->parm[i]);
        }

        printf("\r\nSTATUS REGISTERS\r\n");
        for ( i = 0; i <= 28; i = i + 2 )
        {
                printf("xpc2575->sta[%d] = %x\r\n ",i, XPC2575->sta[i]);
        }
}

/*************************************************************************

/*
**  Functions:          txmpkts()
**                      strcpy()
**
**  Description:        Transmit packets are set up at addresses which
**                      are defined in the tlook elements (transmitter
**                      lookup table). Here is where we force the packets at
**                      the addresses defined in the function init_table().
**
**
**  Memory:             The area defined in the tlook elements, the transmit
**                      data buffers N(s)=0 through N(s)=7.
**
*/

 _txmpkts()
{
unsigned char  buff;


        /* XPC2575 transmit packets */
        strcpy( ((char *) 0xc600), "This is packet 0 from XPC2575");
        strcpy( ((char *) 0xc700), "This is packet 1 from XPC2575");
        strcpy( ((char *) 0xc800), "This is packet 2 from XPC2575");
        strcpy( ((char *) 0xc900), "This is packet 3 from XPC2575");
        strcpy( ((char *) 0xca00), "This is packet 4 from XPC2575");
        strcpy( ((char *) 0xcb00), "This is packet 5 from XPC2575");
        strcpy( ((char *) 0xcc00), "This is packet 6 from XPC2575");
        strcpy( ((char *) 0xcd00), "This is packet 7 from XPC2575");

}
```

```
/*
        ****************************************************************
        *       Initialize the 9519 interrupt controller               *
        ****************************************************************
*/

#include <globs.h>
#include <9519.h>

init9519 ()
{
printf("in 9519.c");
        USR_9519->command = 0x00;       /* reset 9519 */
        USR_9519->command = 0x80;       /* mode reg; interrupt active low */
        USR_9519->command = 0xa9;       /* arm chip */
        USR_9519->command = 0xc0;       /* preselect auto-clear reg.*/
        USR_9519->data =    0xff;       /* clear pending irq */


        USR_9519->command = 0xe4;       /* load vector map ireq 4 */
        USR_9519->data =    0x45;       /* vector for PCBP addr = 0x110 */

        USR_9519->command = 0xe5;       /* load vector map ireq 5 */
        USR_9519->data =    0x44;       /* vector for PCBP addr = 0x110 */

        USR_9519->command = 0xe6;       /* load vector map ireq 6 */
        USR_9519->data =    0x43;       /* vector for PCBP addr = 0x110 */

        USR_9519->command = 0xe7;       /* load vector map ireq 7 */
        USR_9519->data =    0x42;       /* vector for PCBP addr = 0x110 */


        USR_9519->command = 0x40;       /* clear all irr bits. */
        USR_9519->command = 0xb0;       /* presel int. mask reg. */
        USR_9519->data =    0x0f;       /* unmask IREQ4 through IREQ7 */
}

/*
        ****************************************************************
        *       Turn off the 9519 interrupt controller                 *
        ****************************************************************
*/

stop9519 ()
{
        USR_9519->command = 0xb0;       /* presel int. mask reg. */
        USR_9519->data =    0xff;       /* mask all interrupts */
}
```

```
#include <globs.h>

/* define three pointers to the XPC's in an array */
XPC_8 *xpc8[3];

TABLE *tables[3];



main()
{

        /* initialize structures to array of pointers */
        printf("init_structs()\n");
        init_structs();

        /* initialize the ports for the 8255 device */
        printf("init_8255()\n");
        init_8255();

        /* initialize the upper address bits for the XPC-8 */
        printf("init_up_add_bit()\n");
        init_up_add_bit();

        /* place in memory some packets */
        printf("txmpkts()\n");
        txmpkts();

        /* initialize the XPC parameter registers */
        printf("initialize_XPC8()\n");
        initialize_XPC8();

        /* initialize the TLOOK tables */
        printf("init_table()\n;");
        init_table();

        /* initialize 9519 */
        /*printf("init9519()\n");
        init9519();
        */

        /* initialize 2575
        printf("init_2575()\n");
        init_2575();
        */

        /* set brdys bit in xpc for buffer ready */
        printf("set_brdys()\n");
        set_brdys();

        /* set up command register for send */
        printf("init_cmd()\n");
        init_cmd();


        /* dump all status registers on the XPC's */
        printf("dump_sta()\n");
        dump_sta();

        /* set up command register for send */
        printf("init_cmd()\n");
        init_cmd();
```

```
        /* initialize T7102 XPC2575 */
        go_2575();
}
```

```
|  MC68000 monitor modified for XPC Tutor board
|  by Michael R. Kost Jr.
|  August 1986

        .text
putchr = 0xe000
getchr = 0xe004
putchrm = 0xe008
getchrm = 0xe00c

|Modem queues

MXMTHEAD = 0xe010
MXMTTAIL = 0xe014
MRCVHEAD = 0xe018
MRCVTAIL = 0xe01c

|*** terminal queues (T1)

TXMTHEAD = 0xe020
TXMTTAIL = 0xe024
TRCVHEAD = 0xe028
TRCVTAIL = 0xe02c


BUFPTR = 0xe030
intrchar = 0xe034
tslshflg = 0xe035
editflg = 0xe036
lineflg = 0xe037
eolchar = 0xe038
killchar = 0xe039
resetflg = 0xe03a
topstack = 0xe03c

| ************************** N O T E ***********************************
| * Since the C compiler, as presently customized, expects to pick up
| * the vectors to the UPUTCHR, UGETCHR and CHKCHR routines from the low
| * RAM locations of 0x440, 0x444 and 0x448 respectively, we will have to
| * put the locations of those routines into the ROM at 0x440, 0x444
| * and 0x448 "manually", and the following 3 RAM vectors will not be
| * used by the C compiler to find the routines.
| * The ROM code will have to start at a higher address, 0x500.
| * -- mrk 8/7/86
| ******************************************************************************

|C compiler vectors not assigned here but in ROM Vectors
|UPUTCHR = 0x440 **** uputc:
|UGETCHR = 0x444 **** ugetc:
|CHKCHR = 0x448

pausflg = 0xe050        | Flag set when Cntl-S typed to pause output
rawflag = 0xe051        | Set to 1 when no local Cntl-S/Q control
exformt = 0xe052        | set when loading/saving in extended address
                        | Intel format
buffer = 0xe600


|*** MODEM buffers

MRCVBUF = 0xe400
MXMTBUF = 0xe500


|*** TERMINAL 0 buffers
```

```
TRCVBUF = 0xe200
TXMTBUF = 0xe300


KEYBUF = 0xe600
monref = notfnd-2

|*************************************************************************
|* end of defines
|*************************************************************************

|*************************************************************************
|The assembler from Murry Hill(a68)(in $HOME/assembler/68k) is a relative
|assembler therefore we must find where the address of our interrupt
|routines are by executing this code then searching the .data section
|for these interrupt service routines so we can load these addresses
|into the proper vector location in ROM. These are here to remind us
|to which service routine we need to attach to which vector.
|This is code that NEVER should(or could) be executed.
|*************************************************************************


|These vectors will be in ROM on the XPC Tutor, be sure to put in!
|       lea pc@(buserr),a0
|       movl a0,0x000008
|       lea pc@(addrerr),a0
|       movl a0,0x00000C
|       lea pc@(illegal),a0
|       movl a0,0x000010
|       lea pc@(divzero),a0
|       movl a0,0x000014
|       lea pc@(uninit),a0
|       movl a0,0x00003c
|       lea pc@(duartlint),a0
|       movl a0,0x000100          |First "user" interrupt vector, No. 40
|*** use when install 2nd duart
|       lea pc@(duart2int),a0
|       movl a0,0x000104          |Second "user" interrupt vector, No. $44
|*** End of interrupts
| These vectors are in ROM on the so we don't need this code
|       lea pc@(** outchar ** should be uputc ***),a0
|       movl a0,UPUTCHR
|       lea pc@(??????inchar),a0
|       movl a0,UGETCHR
|       lea pc@(chkc),a0
|       movl a0,CHKCHR
|*** END of ROM vectors ***

|*************************************************************************
|*************************************************************************
|       ********        START OF ROM            *******
|*************************************************************************
|*************************************************************************

MON1.0: orw #0x0700,sr          | Disable interrupts if doing software reset
        movb #3,intrchar        | set default interrupt char to Cntl-C
        clrb tslshflg
        clrb pausflg
        clrb rawflag
        movl #TRCVBUF,d0
        movl d0,TRCVTAIL
        movl d0,TRCVHEAD
        movl #TXMTBUF,d0
        movl d0,TXMTTAIL
```

```
        movl d0,TXMTHEAD
        movl #MRCVBUF,d0
        movl d0,MRCVTAIL
        movl d0,MRCVHEAD
        movl #MXMTBUF,d0
        movl d0,MXMTTAIL
        movl d0,MXMTHEAD
        movl #0x20000,a0          | where RAM starts
ramloop: movb a0@,d1              | Save byte there now
        movb #0x55,a0@
        cmpb #0x55,a0@
        bnes endfound
        movb #0xaa,a0@
        cmpb #0xaa,a0@
        bnes endfound
        movb d1,a0@+
        bras ramloop
endfound: movl a0,a7              | Initialize stack pointer
        movl a0,topstack          | Save top of stack pointer
        bsr initport              | Initialize DUARTs
        movb #0x55,resetflg       | If flag not 55 then interrupt cold starts
        movb #0x18,killchar       | Set default kill char to Cntl-X
        movb #0x0d,eolchar        | Set default end of line char to <CR>
        lea pc@(putc),a5
        movl a5,putchr
        clrb tslshflg
        clrb pausflg
        clrb rawflag
        lea pc@(sysmsg),a0
        bsr crlfmsg
mainst: lea pc@(putcm),a3
        movl a3,putchrm
        lea pc@(getcm),a4
        movl a4,getchrm
        lea pc@(putc),a5
        movl a5,putchr
        lea pc@(getc),a6
        movl a6,getchr
        movb #0xff,editflg
        clrb lineflg
        clrb pausflg
        clrb rawflag
        cmpb #0x55,resetflg       | Check to see if topstack probably valid
        bne MON1.0                | Do total reset if system RAM was overwritte
        movl topstack,a7          | Reinitialize stack pointer
        movw sr,d0
        andw #0xf8ff,d0           | Mask out interrupt level bits
        movw d0,sr                | Enable interrupts, processor at priority 0
mainlp: lea pc@(prmptmsg),a0
        bsr crlfmsg
        movl #buffer,a0
        movw #0xff,d1
        bsr getbuf
        bsrs intrpt
        bras mainlp
intrpt: movb a0@+,d0
        bnes init
        rts
init:   bsr upconv
        lea pc@(tblend),a2
        lea pc@(cmndtbl),a1
search: cmpb a1@,d0
        beqs found
        cmpl a2,a1
        bccs notfnd
        addql #6,a1
```

```
        bras search
found:  addql #2,a1
        movl a1@,a1
        jmp pc@(0,a1:L)
notfnd: lea pc@(ntfndmsg),a0
        bsr msg
        rts
gethex: moveq #0,d2         | init number
        moveq #8,d1         | digit count
dlmeat: movb a0@+,d0
        bsrs chkdlm
        beqs dlmeat
        tstb d0
        bnes gtnxtnm
        andb #0xfb,cc
        rts
gtnxtnm: bsrs valdhex
        bnes dlmchk
        lsll #4,d2
        addb d0,d2
        subqw #1,d1
        bmis hexdone
        movb a0@+,d0
        bras gtnxtnm
dlmchk: bsrs endchk
hexdone: rts
endchk: tstb d0
        beqs end
chkdlm: cmpb #0x20,d0
        beqs end
        cmpb #0x2c,d0
end:    rts
valdhex: cmpb #0x30,d0
        blts nogood
        cmpb #0x39,d0
        bgts chkaf
        subb #0x30,d0
goodrt: orb #4,cc
        rts
chkaf:  bsrs upconv
        cmpb #0x41,d0
        blts nogood
        cmpb #0x46,d0
        bgts nogood
        subb #0x37,d0
        bras goodrt
nogood: andb #0xfb,cc
        rts
upconv: cmpb #0x61,d0
        bcss noconv
        cmpb #0x7b,d0
        bccs noconv
        andb #0xdf,d0
noconv: rts
crlfmsg: bsr prnl
msg:    movb a0@+,d0
        beqs msgend
        jsr a5@
        bras msg
msgend: rts
getbuf: moveml #0xc080,sp@-
chrloop: jsr a6@                | get a char.
        cmpb #8,d0
        beqs backsp
        jsr a5@                 | put char.
        cmpb #0x18,d0
```

```
        bnes chkcr
erasln: cmpw sp@(6),d1
        beqs chrloop
        movb #8,d0
        jsr a5@
        addqw #1,d1
        subql #1,a0
        bras erasln
chkcr:  cmpb #0x0d,d0
        bnes chkctl
        movb #0x0a,d0
        jsr a5@
        clrb a0@
exit:   moveml sp@+,#0x0103
        rts
chkctl: cmpb #0x1f,d0
        blss chrloop
        cmpb #0x7f,d0
        bccs chrloop
        cmpb #0x5c,d0
        bnes stochr
        jsr a6@
        jsr a5@
stochr: movb d0,a0@+
        subqw #1,d1
        bnes chrloop
        orb #1,cc
        bras exit
backsp: cmpw sp@(6),d1
        beqs chrloop
        jsr a5@
        addqw #1,d1
        subql #1,a0
        bras chrloop
prbyte: movw sr,sp@-            | prints byte in d0
        movw d0,sp@-
        lsrb #4,d0
        bsrs itoa
        jsr a5@
        movb sp@(1),d0
        bsrs itoa
        jsr a5@
        movw sp@+,d0
        movw sp@+,sr
        rts
prword: movw sr,sp@-            | Prints word in d0
        movl d0,sp@-            | save d0
        lsrw #8,d0
        bsrs prbyte
        movl sp@,d0
        bsrs prbyte
        movl sp@+,d0
        movw sp@+,sr
        rts
prlong: movw sr,sp@-            | Prints longword in a0
        movl d0,sp@-
        movl a0,d0
        swap d0
        bsrs prword
        swap d0
        bsrs prword
        movl sp@+,d0
        movw sp@+,sr
        rts
itoa:   andb #0x0f,d0
        cmpb #9,d0
```

```
        bhis afconv
        addb #0x30,d0
        rts
afconv: addb #0x37,d0
        rts
pr2sp:  bsr pr1sp
pr1sp:  movw sr,sp@-
        movl d0,sp@-
        movb #0x20,d0
        jsr a5@
        movl sp@+,d0
        movw sp@+,sr
        rts
prnsp:  tstb d0
        beqs prdone
        bsrs pr1sp
        subqb #1,d0
        bras prnsp
prdone: rts
prnl:   movw sr,sp@-
        movl d0,sp@-
        movb #0x0d,d0
        jsr a5@
        movb #0x0a,d0
        jsr a5@
        movl sp@+,d0
        movw sp@+,sr
        rts
pradm:  bsrs pr1ong
        bsrs pr2sp
        movb a0@,d0
        bsr prbyte
        bsrs pr2sp
        rts
prcntrl: movw sr,sp@-
        movw d0,sp@-
        andb #0x7f,d0
        cmpb #0x20,d0
        bccs prnrmal
        movb #0x5e,d0
        jsr a5@
        movb sp@(1),d0
        orb #0x40,d0
prnt:   jsr a5@
        movw sp@+,d0
        movw sp@+,sr
        rts
prnrmal: bsrs pr1sp
        bras prnt
lstfld: bsr gethex
        bnes lstend
findennd: bsr chkdlm
        bnes tstend
        movb a0@+,d0
        bras findennd
tstend: tstb d0
lstend: rts
exam:   movw sr,sp@-
        moveml #0xe080,sp@-
        bsrs lstfld
        beqs getadr
        lea pc@(bargmsg),a0
        bsr msg
        bsr errmsg
        bra edone
getadr: movl d2,a0
```

```
prnladr: bsr prnl
pradr:   bsrs pradm
         bsrs prcntrl
         bsr pr2sp
         clrl d1
         clrb d2
getnxt:  jsr a6@
         cmpb #0x0d,d0
         bnes hexchk
stochk:  tstb d2
         beqs incadr
         movb d1,a0@
incadr:  addql #1,a0
         bras prnladr
hexchk:  jsr a5@
         bsr valdhex
         bnes spcchk
         lslw #4,d1
         addb d0,d1
         addqb #1,d2
         bras getnxt
spcchk:  cmpb #0x20,d0
         beqs stochk
         cmpb #0x2d,d0
         bnes slshchk
         tstb d2
         beqs decadr
         movb d1,a0@
decadr:  subql #1,a0
         bras prnladr
slshchk:  cmpb #0x2f,d0
         bnes qotchk
         tstb d2
         beqs prnladr
         movb d1,a0@
         bras prnladr
qotchk:  cmpb #0x22,d0
         bnes quitchk
         tstb d2
         bnes errout
entchrs:  clrw d1
         bsr getbuf
         bccs findend
         addl #0x10000,a0
         bras entchrs
findend:  movb a0@+,d0
         bnes findend
         bra prnladr
quitchk:  cmpb #0x51,d0
         bnes errout
edone:   moveml sp@+,#0x0107
         movw sp@+,sr
         rts
errout:  bsrs errmsg
         bra prnladr
go:      bsr lstfld
         bnes badarg
         btst #0,d2
         bnes badarg
         movl d2,a0
         jmp a0@
badarg:  lea pc@(bargmsg),a0
         bsr msg
         bsrs errmsg
         rts
errmsg:  movw sr,sp@-
```

```
                movem1 #0x8080,sp@-
                lea pc@(error),a0
                bsr crlfmsg
                movem1 sp@+,#0x0101
                movw sp@+,sr
                rts
gt2args: bsr gethex
                bnes gt2don
                movl d2,d3
                tstb d0
                beqs bad2
                movb a0@,d0
                cmpb #0x23,d0
                bnes get2
                addql #1,a0
                bsr gethex
                bnes gt2don
                movl d2,d4
good2:  orb #4,cc
gt2don: rts
get2:   bsr gethex
                bnes bad2
                subl d3,d2
                blss bad2
                movl d2,d4
                addql #1,d4
                bras good2
bad2:   andb #0xfb,cc
                rts
write:  movb a0@,d0
                bsr upconv              | convert to uppercase
                cmpb #0x58,d0           | is char a 'X' ?
                bnes stdwrit            | no, then do a standard Intel format write
                movb #0xff,exformt      | yes, set extended format flag
                addql #1,a0             | move pointer past the 'X'
                bras wrtargs
stdwrit: clrb exformt           | clear the extended format flag
wrtargs: bsrs gt2args           | start -> d3, length -> d4
                beqs stwrt
                bsr badarg
                rts
stwrt:  exg a3,a5
                movb #0x02,0x200015     | Disable Channel B (modem) receiver
wrtloop:movl d4,d2              | put length in d2
                cmpl #16,d2             | is length > 16 bytes ?
                blss wrtline            | if <= 16 just use length as record length
                movl #16,d2             | else record length will be 16 bytes
wrtline: movb #0x3a,d0
                jsr a5@
                movb d2,d1              | start checksum
                movb d2,d0
                bsr prbyte
                movl d3,d0
                tstb exformt
                beqs lowaddr
                swap d0
                lsrw #8,d0
                bsr prbyte
                addb d0,d1              | add in to checksum
                movl d3,d0
                swap d0
                bsr prbyte
                addb d0,d1              | add in to checksum
                movl d3,d0
lowaddr: lsrw #8,d0
                bsr prbyte
```

```
        addb d0,d1
        movw d3,d0
        bsr prbyte
        addb d0,d1
        clrb d0
        bsr prbyte
        movl d3,a0
bytloop: movb a0@+,d0
        bsr prbyte
        addb d0,d1
        subqb #1,d2
        bnes bytloop
        movl a0,d3
        negb d1
        movb d1,d0
        bsr prbyte
        movb #0x0d,d0
        jsr a5@
        movb #0x2a,d0
        jsr a3@
        subl #16,d4
        bhis wrtloop
        movb #0x3a,d0           | last line when it falls thru to here
        jsr a5@
        movb #0,d0
        movb #6,d2
outzero: bsr prbyte
        subqb #1,d2
        bnes outzero
        movb #0x0d,d0
        jsr a5@
        movb #4,d0
        jsr a5@
        movb #0x2a,d0
        jsr a3@
        exg a3,a5
        movb #0x01,0x2000015    | Enable Channel B (modem) receiver
        rts
load:   movb a0@,d0
        bsr upconv              | convert char to uppercase
        cmpb #0x58,d0           | is it an 'X' ?
        bnes stdload            | no, just load standard Intel format
        movb #0xff,exformt      | yes, set extended address format flag
        addql #1,a0             | move pointer past the 'X'
        bras getoffs
stdload: clrb exformt
getoffs: bsr lstfld
        beqs offset
        tstb d0
        bne badarg
        clrl d2
offset: movl #0,a1
        movb #0x0d,d0
        jsr a3@
findbgn: jsr a4@
        cmpb #0x3a,d0
        beqs fndbgn
        cmpb #0x30,d0
        bne findbgn
badload: lea pc@(bdldmsg),a0
prmsgad: bsr crlfmsg
        movl a1,a0
        bsr prlong
lend:   rts
fndbgn: clrb d1                 | init checksum
        bsrs getbyte            | get first byte which is record length
```

```
            tstb d0                 | is length of this record == 0 ?
            beqs lend               | yes, then end the load
            movb d0,d3              | number of bytes to load in this line
            clrl d0
            bsrs getbyte
            lslw #8,d0
            movl d0,a1
            clrw d0
            bsrs getbyte
            addl d0,a1
            tstb exformt            | are we supposed to look for extended
                                    | address format ?
            beqs addoffs            | no, then go add the offset to the address
            movl a1,d0              | add above left result in a1, not d0
            swap d0                 | first 16 bits of address are now in high par
            clrw d0                 | init to get next byte
            bsrs getbyte            | get third byte of address
            lslw #8,d0              | put third byte in correct position
            movl d0,a1              | save address with 3 bytes in a1
            clrl d0                 | init to get last byte
            bsrs getbyte            | get fourth byte of address
            addl d0,a1              | add in fourth byte to other 3 in a1
addoffs:    addl d2,a1              | add the offset
            bsrs getbyte
            tstb d0
            bnes badtype
getnxtb:    bsrs getbyte
            movb d0,a1@
            cmpb a1@+,d0
            bnes badload
            subqb #1,d3
            bnes getnxtb
            bsrs getbyte
            bnes bdchksm
            movb #0x2a,d0
            jsr a5@
            bras findbgn
bdchksm:    lea pc@(chksmsg),a0
            bras prmsgad
badtype:    lea pc@(typemsg),a0
            bsr crlfmsg
            rts
getbyte:    jsr a4@
            bsr valdhex
            bnes berr
            movb d0,d4
            jsr a4@
            bsr valdhex
            bnes berr
            lslb #4,d4
            addb d4,d0
            addb d0,d1              | add in to checksum
            rts
berr:       addql #4,sp
            lea pc@(bdbytmsg),a0
            bra prmsgad
mov:        bsr gt2args             | start -> d3, length -> d4
            beqs gtadr2
mexit:      bsr badarg
            rts
gtadr2:     bsr lstfld              | get "to-address" in d2
            bnes mexit
            movl d3,a0
            movl d2,a1
            cmpl a0,a1              | dest : start
            bcss mvback
```

```
        addl d4,a0
        addl d4,a1
mvloop: movb a0@-,a1@-
        subql #1,d4
        bnes mvloop
        rts
mvback: movb a0@+,a1@+
        subql #1,d4
        bnes mvback
        rts
disasmbl: bsr lstfld
        bne badarg
        movl d2,a2                  | load a2 with address to start disassembly
disloop: bsrs disasm
        jsr a6@                     | get a char from the monitor
        bsr upconv
        cmpb #0x51,d0
        bnes disloop
        rts
disasm: moveml #0xf8d2,a7@-         | save d0,d1,d2,d3,d4,a0,a1,a3,a6 on stack
        subl #128,a7
        movl a7,a6                  | keep pointer to variable storage stack space
        movl a7,a1
        movw #79,d0
        bsr prnspc                  | prints @ a1 and inc's it
        clrb a1@                    | mark end of buffer
        movl a7,a1
        movl a2,d0
        bsr prlongd0                | prints d0 as long to buffer @ a1
        movw #2,d0
        bsr prnspc
        lea a7@(38),a3              | a3 will point to opcode field
        movw a2@+,d0
        movw d0,d1                  | d1 holds opcode
        bsr prworddd0               | prints d0 as word to buffer @ a1
        movw #1,d0
        bsr prnspc
| a1 now set up to print other words of this instruction
        bsrs propcode
        movl a7,a1
outloop: movb a1@+,d0
        beqs disdone
        jsr a5@
        bras outloop
disdone: movb #0x0d,d0
        jsr a5@
        movb #0x0a,d0
        jsr a5@
        addl #128,sp
        moveml a7@+,#0x4b1f         | restore a6,a3,a1,a0,d4,d3,d2,d1,d0 from stac
        rts
propcode: movw d1,d2
        rolw #4,d2
        andw #0x0f,d2               | d2 now has upper nibble
        bne cmp1
        cmpw #0x003c,d1
        bne orsr
        lea pc@(ormsg),a0
ccimm:  bsr dmsg
        bsr dotb
        bsrs prbytimm
        lea pc@(ccmsg),a0
        bsr dmsg
        rts
prbytimm: bsr spnumdol
        movw a2@+,d0
```

```
                bsr prworddO
                movb #Ox2O,a1@+          | ' '
                exg a1,a3
                bsr prbytedO
                exg a1,a3
                movb #Ox2c,a3@+
                rts
prwdimm:        bsr spnumdo1
                bsr wordimm
                movb #Ox2c,a3@+
                rts
orsr:           cmpw #Ox007c,d1
                bnes andcc
                lea pc@(ormsg),aO
srimm:          bsr dmsg
                bsr dotw
                bsrs prwdimm
                lea   pc@(srmsg),aO
                bsr dmsg
                rts
andcc:          cmpw #Ox023c,d1
                bnes andsr
                lea pc@(andmsg),aO
                bra ccimm
andsr:          cmpw #Ox027c,d1
                bnes eorcc
                lea pc@(andmsg),aO
                bras srimm
eorcc:          cmpw #Ox0a3c,d1
                bnes eorsr
                lea pc@(eormsg),aO
                bra ccimm
eorsr:          cmpw #Ox0a7c,d1
                bnes chkbit8
                lea pc@(eormsg),aO
                bras srimm
chkbit8:        btst #8,d1
                beq otherO
                movw d1,d3
                andw #Ox0038,d3          | get bits 5,4,3
                cmpw #8,d3               | = 001 ?
                bne dynambit
                movw d1,d3
                andw #Ox00cO,d3          | get bits 7,6
                bnes mpO1
                lea pc@(movepmsg),aO                    —
                bsr dmsg
                bsr dotw
prdisp:         movb #Ox20,a3@+          | ' '
                bsr dispad
                movb #Ox2c,a3@+          | ','
                bsrs d11109              | print data reg in bits 11,10,9
                rts
a11109:         movb #Ox41,a3@+          | 'A'
                bras reg11109
d11109:         movb #Ox44,a3@+          | 'D'
reg11109:       movw d1,d3
                rolw #7,d3
                andw #Ox0007,d3
                addb #Ox30,d3
                movb d3,a3@+
                rts
mpO1:           cmpw #Ox0040,d3
                bnes mpO2
                lea pc@(movepmsg),aO
                bsr dmsg
lea pc@(movepmsg),aO
bsr dmsg
```

```
        bsr dotl
        bras prdisp
mp02:   cmpw #0x0080,d3
        bnes mp03
        lea pc@(movepmsg),a0
        bsr dmsg
        bsr dotw
        movb #0x20,a3@+         | ' '
pd0:    bsr d11109
        movb #0x2c,a3@+         | ','
        bsr dispad
        rts
mp03:   lea pc@(movepmsg),a0
        bsr dmsg
        bsr dotl
        bras pd0
dynambit:
        movb #0x42,a3@+         | 'B'
        movw d1,d3
        andw #0x00c0,d3         | get bits 7,6
        bnes dyn1
        lea pc@(tstmsg),a0
bitaddr: bsr dmsg
        movb #0x20,a3@+         | ' '
        bsr d11109
        movb #0x2c,a3@+         | ','
        bsr prea
        rts
dyn1:   cmpw #0x0040,d3
        bnes dyn2
        lea pc@(chgmsg),a0
        bras bitaddr
dyn2:   cmpw #0x0080,d3
        bnes dyn3
        lea pc@(clrmsg),a0
        bras bitaddr
dyn3:   lea pc@(setmsg),a0
        bras bitaddr
other0: movw d1,d3
        andw #0x0f00,d3
        cmpw #0x0800,d3
        bnes tryimm0
        movb #0x42,a3@+         | 'B'
        movw d1,d3
        andw #0x00c0,d3
        bnes stat1
        lea pc@(tstmsg),a0
bitstat: bsr dmsg
        bsr prwdimm
        bsr prea
        rts
stat1:  cmpw #0x0040,d3
        bnes stat2
        lea pc@(chgmsg),a0
        bras bitstat
stat2:  cmpw #0x0080,d3
        bnes stat3
        lea pc@(clrmsg),a0
        bras bitstat
stat3:  lea pc@(setmsg),a0
        bras bitstat
tryimm0: cmpw #0x0000,d3
        bnes andimm
        lea pc@(ormsg),a0
immea:  bsr dmsg
        bsr prsize              | use bits 6,7 to print out .B .W or .L
```

```
        bsr primm
        bsr prea
        rts
andimm: cmpw #0x0200,d3
        bnes subimm
        lea pc@(andmsg),a0
        bras immea
subimm: cmpw #0x0400,d3
        bnes addimm
        lea pc@(submsg),a0
        bras immea
addimm: cmpw #0x0600,d3
        bnes eorimm
        lea pc@(addmsg),a0
        bras immea
eorimm: cmpw #0x0a00,d3
        bnes cmpimm
        lea pc@(eormsg),a0
        bras immea
cmpimm: cmpw #0x0c00,d3
        bne badop
        lea pc@(cmpmsg),a0
        bras immea
cmp1:   cmpb #1,d2              | see if upper nibble = 1
        bnes cmp2
        lea pc@(movmsg),a0
        bsr dmsg
        bsr dotb
        movb #0,a6@(127)        | set size flag to 0 for byte
movpr:  movb #0x20,a3@+
        bsr prea
        movb #0x2c,a3@+
        bsr prmvdst
        rts
cmp2:   cmpb #2,d2
        bnes cmp3
        lea pc@(movmsg),a0
        bsr dmsg
        bsr dotl
        movb #2,a6@(127)        | set size flag to 2 for long
        bras movpr
cmp3:   cmpb #3,d2
        bnes cmp4
        lea pc@(movmsg),a0
        bsr dmsg
        bsr dotw
        movb #1,a6@(127)        | set size flag to 1 for word
        bras movpr
cmp4:   cmpb #4,d2
        bne cmp5
        movw d1,d3
        andw #0x0f00,d3
        bnes tryclr
| Here if was 0 - for NEGX or Move from SR
        movw d1,d3
        andw #0x00c0,d3
        cmpw #0x00c0,d3
        bnes negx
        lea pc@(movfsr),a0
        bsr dmsg
        bsr prea
        rts
negx:   lea pc@(negxmsg),a0
        bsr prsizea
        rts
tryclr: cmpw #0x0200,d3
```

```
        bnes tryneg
        lea pc@(clrmsg),a0
        bsr prsizea
        rts
tryneg: cmpw #0x0400,d3
        bnes trynot
        movw d1,d3
        andw #0x00c0,d3
        cmpw #0x00c0,d3
        bnes neg
| Here if Move to CCR instruction
        lea pc@(movmsg),a0
        bsr dmsg
        bsr dotw
        movb #0x20,a3@+          | ' '
        bsr prea
        movb #0x2c,a3@+          | ','
        lea pc@(ccmsg),a0
        bsr dmsg
        rts
neg:    lea pc@(negmsg),a0
        bsr prsizea
        rts
trynot: cmpw #0x0600,d3
        bnes trynbcd
        movw d1,d3
        andw #0x00c0,d3
        cmpw #0x00c0,d3
        bnes not
| Here if Move to SR instruction
        lea pc@(movmsg),a0
        bsr dmsg
        bsr dotw
        movb #0x20,a3@+          | ' '
        bsr prea
        movb #0x2c,a3@+          | ','
        lea pc@(srmsg),a0
        bsr dmsg
        rts
not:    lea pc@(notmsg),a0
        bsr prsizea
        rts
trynbcd:cmpw #0x0800,d3
        bne trytst
        movw d1,d3
        andw #0x00f8,d3
        cmpw #0x0040,d3
        bnes tryextw
        lea pc@(swapmsg),a0
        bsr dmsg
spcdreg: movb #0x20,a3@+                 | ' '
dreg:   movb #0x44,a3@+          | 'D'
        bsr prregnum
        rts
tryextw: cmpw #0x0080,d3
        bnes tryextl
        lea pc@(extmsg),a0
        bsr dmsg
        bsr dotw
        bras spcdreg
tryextl: cmpw #0x00c0,d3
        bnes seenbcd
        lea pc@(extmsg),a0
        bsr dmsg
        bsr dotl
        bras spcdreg
```

```
seenbcd: andw #0x00c0,d3
         bnes trypea
         lea pc@(nbcdmsg),a0
mspcea: bsr dmsg
         movb #0x20,a3@+
         bsr prea
         rts
trypea: cmpw #0x0040,d3
         bnes movtoea
         lea pc@(peamsg),a0
         bras mspcea
movtoea: cmpw #0x0080,d3
         bnes movtoeal
         lea pc@(movemwms),a0
movto:  bsr dmsg
         movb #0x20,a3@+          | ' '
         bsr preglist
         bra commaea
movtoeal: cmpw #0x00c0,d3
         bne badop
         lea pc@(movemlms),a0
         bras movto
trytst: cmpw #0x0a00,d3
         bnes seemear
         movw d1,d3
         andw #0x00c0,d3
         cmpw #0x00c0,d3
         beqs seeill
         lea pc@(tstmsg),a0
         bsr dmsg
         bsr prsizea
         rts
seeill: cmpw #0x4afc,d1
         bnes trytas
         lea pc@(illmsg),a0
         bsr dmsg
         rts
trytas: lea pc@(tasmsg),a0
         bra mspcea
seemear: movw d1,d3
         andw #0xff80,d3
         cmpw #0x4c80,d3
         bnes trapgrp
         btst #6,d1
         bnes mvfrmeal
         lea pc@(movemwms),a0
movfrm: bsr dmsg
         movb #0x20,a3@+          | ' '
         movw a2@+,d0
         bsr prworddO             | print out register list word first
         movb #0x20,a1@+          | ' '
         movw d0,d4               | save register list
         bsr prea                 | EA word follows register list word
         movb #0x2c,a3@+          | ','
         movw d4,d0               | get register list back in d0
         bsr regsonly             | print out register list using current d0
         rts
mvfrmeal: lea pc@(movemlms),a0
         bras movfrm
trapgrp: andw #0x0f00,d3
         cmpw #0x0e00,d3
         bne seechk
         cmpw #0x4e70,d1
         bnes chktrap
         lea pc@(resetmsg),a0
         bsr dmsg
```

```
        rts
chktrap: movw d1,d3
        andw #0xfff0,d3
        cmpw #0x4e40,d3
        bnes trylink
        lea pc@(trapmsg),a0
        bsr dmsg
        movb #0x20,a3@+          | ' '
        movb #0x23,a3@+          | '#'
        movw d1,d3
        andw #0x000f,d3
        cmpb #9,d3
        bles digit2
        movb #0x31,a3@+          | '1'
        subb #10,d3
digit2: addb #0x30,d3
        movb d3,a3@+
        rts
trylink: movw d1,d3
        andw #0x00f8,d3
        cmpw #0x0050,d3
        bnes seeunlk
        lea pc@(linkmsg),a0
        bsr dmsg
        bsrs spcAreg
        movb #0x2c,a3@+          | ','
immop2: movb #0x23,a3@+          | '#'
        movb #0x24,a3@+          | '$'
wordimm: movw a2@+,d0
        bsr prwordd0
        movb #0x20,a1@+          | ' '
        exg a1,a3
        bsr prwordd0
        exg a1,a3
        rts
seeunlk: cmpw #0x0058,d3
        bnes seemovu
        lea pc@(unlkmsg),a0
        bsr dmsg
spcAreg: movb #0x20,a3@+         | ' '
prAreg: movb #0x41,a3@+          | 'A'
        bsr prregnum
        rts
seemovu: cmpw #0x0060,d3
        bnes movfrmu
        lea pc@(movmsg),a0
        bsr dmsg
        bsr dotl
        bsrs spcAreg
        movb #0x2c,a3@+          | ','
        lea pc@(uspmsg),a0
        bsr dmsg
        rts
movfrmu: cmpw #0x0068,d3
        bnes seenop
        lea pc@(movmsg),a0
        bsr dmsg
        bsr dotl
        movb #0x20,a3@+          | ' '
        lea pc@(uspmsg),a0
        bsr dmsg
        movb #0x2c,a3@+          | ','
        bras prAreg
seenop: andw #0x00f0,d3
        cmpw #0x0070,d3
        bne seejsr
```

```
        cmpw #0x4e71,d1
        bnes seestop
        lea pc@(nopmsg),a0
        bsr dmsg
        rts
seestop: cmpw #0x4e72,d1
        bnes seerte
        lea pc@(stopmsg),a0
        bsr dmsg
        movb #0x20,a3@+          | ' '
        bra immop2
seerte: cmpw #0x4e73,d1
        bnes seerts
        lea pc@(rtemsg),a0
        bsr dmsg
        rts
seerts: cmpw #0x4e75,d1
        bnes seetrapv
        lea pc@(rtsmsg),a0
        bsr dmsg
        rts
seetrapv: cmpw #0x4e76,d1
        bnes seertr
        lea pc@(trapvmsg),a0
        bsr dmsg
        rts
seertr: cmpw #0x4e77,d1
        bne badop
        lea pc@(rtrmsg),a0
        bsr dmsg
        rts
seejsr: movw d1,d3
        andw #0x00c0,d3
        cmpw #0x0080,d3
        bnes seejmp
        lea pc@(jsrmsg),a0
        bra mspcea
seejmp: cmpw #0x00c0,d3
        bne badop
        lea pc@(jmpmsg),a0
        bra mspcea
seechk: movw d1,d3
        andw #0x01c0,d3
        cmpw #0x0180,d3
        bnes chklea
        lea pc@(chkmsg),a0
        bsr mspcea
        bra comaD
chklea: cmpw #0x01c0,d3
        bne badop
        lea pc@(leamsg),a0
        bsr mspcea
        bra comaA
cmp5:   cmpb #5,d2
        bne cmp6
        movw d1,d3
        andw #0x00c0,d3
        cmpw #0x00c0,d3
        beqs setcond
        btst #8,d1
        bnes dosubq
        lea pc@(addmsg),a0
addsubq: bsr dmsg
        movb #0x51,a3@+          | 'Q'
        bsr prsize
        bsrs imm1to8
```

```
commaea: movb #0x2c,a3@+          | ','
         bsr prea
         rts
immlto8: movb #0x20,a3@+          | ' '
         movb #0x23,a3@+          | '#'
         movw d1,d3
         andw #0x0e00,d3
         beqs eightq
         bra reg11109
eightq:  movb #0x38,a3@+          | '8'
         rts
dosubq:  lea pc@(submsg),a0
         bras addsubq
setcond: movw d1,d3
         andw #0x0038,d3
         cmpw #0x0008,d3
         beqs dodbcc
         movb #0x53,a3@+          | 'S'
condea:  bsr prcond
         movb #0x20,a3@+          | ' '
         bsr prea
         rts
dodbcc:  movb #0x44,a3@+          | 'D'
         movb #0x42,a3@+          | 'B'
         movw d1,d3
         andw #0x0f00,d3
         bnes seecond1
         lea pc@(brnmsg),a0
         bras dbccmsg
seecond1: cmpw #0x0100,d3
         bnes regcond
         lea pc@(bramsg),a0
dbccmsg: bsr dmsg
         bras enddbcc
regcond: bsr prcond
enddbcc: bsr spcdreg
         movb #0x2c,a3@+          | ','
         bras longdisp            | Calculate address from 16 bit displacement
cmp6:    cmpb #6,d2
         bnes cmp7
         movw d1,d3
         andw #0x0f00,d3
         bnes seebsr
         lea pc@(bramsg),a0
         bsr dmsg
branch:  movw d1,d0
         andl #0x00ff,d0          | Check displacement
         beqs longbr
         bsr dots
         movb #0x20,a3@+          | ' '
         movb #0x5b,a3@+          | '['
         movb #0x24,a3@+          | '$'
         extw d0                  | extend byte to word
         extl d0                  | extend to long value
         addl a2,d0
         exg a1,a3
         bsr prlongd0
         exg a1,a3
         movb #0x5d,a3@+          | ']'
         rts                      | ' '
longbr:  movb #0x20,a3@+          | '['
longdisp: movb #0x5b,a3@+         | '$'
         movb #0x24,a3@+
         movw a2@,d0
         extl d0
         addl a2,d0
```

```
        exg a1,a3
        bsr prlongd0
        exg a1,a3
        movb #0x5d,a3@+         | ']'
        movw a2@+,d0
        bsr prworddO
        rts
seebsr: cmpw #0x0100,d3
        bnes brcond
        lea pc@(bsrmsg),a0
        bsr dmsg
        bras branch
brcond: movb #0x42,a3@+         | 'B'
        bsr prcond
        bras branch
cmp7:   cmpb #7,d2
        bne cmp8
        btst #8,d1
        bne badop               | MOVEQ instruction has bit 8 clear
        lea pc@(moveqmsg),a0
        bsr dmsg
        movw d1,d0
        exg a1,a3
        bsr prbyted0
        exg a1,a3
comaD:  movb #0x2c,a3@+         | ','
        bra d11109
cmp8:   cmpb #8,d2
        bne cmp9
        movw d1,d3
        andw #0x01c0,d3
        cmpw #0x00c0,d3
        beq dodivu
        cmpw #0x01c0,d3
        beq dodivs
        cmpw #0x0100,d3
        bnes door
        movw d1,d3
        andw #0x0030,d3
        beq dosbcd
door:   lea pc@(ormsg),a0
opmode: bsr dmsg
        movw d1,d3
        andw #0x01c0,d3         | Get op-mode
        bnes chkor1
        bsr dotb
        movb #0,a6@(127)        | set size flag to byte
orea:   movb #0x20,a3@+         | ' '
        bsr prea
        bras comaD
chkor1: cmpw #0x0040,d3
        bnes chkor2
        bsr dotw
        movb #1,a6@(127)        | set size flag to word
        bras orea
chkor2: cmpw #0x0080,d3
        bnes chkor4
        bsr dotl
        movb #2,a6@(127)        | set size flag to long
        bras orea
chkor4: cmpw #0x0100,d3
        bnes chkor5
        bsr dotb
        movb #0,a6@(127)        | set size flag to byte
ordnea: movb #0x20,a3@+         | ' '
        bsr d11109
```

```
        bra commaea
chkor5: cmpw #0x0140,d3
        bnes chkor6
        bsr dotw
        movb #1,a6@(127)        | set size flag to word
        bras ordnea
chkor6: cmpw #0x0180,d3
        bne badop               | This instruction not needed for OR only oth
        bsr dotl
        movb #2,a6@(127)        | set size flag to long
        bras ordnea
dodivu: lea pc@(divumsg),a0
msgeadn: bsr mspcea
        bra comaD
dodivs: lea pc@(divsmsg),a0
        bras msgeadn
dosbcd: lea pc@(sbcdmsg),a0
        bsr dmsg
dpredec: movb #0x20,a3@+        | ' '
        btst #3,d1
        bnes sbcdadd            | uses predecrement addresssing mode if 1
        bsr dreg
        bra comaD
sbcdadd: lea pc@(negparA),a0
        bsr dmsg
        bsr prregnum
        movb #0x29,a3@+         | ')'
        movb #0x2c,a3@+         | ','
        lea pc@(negparA),a0
        bsr dmsg
        bsr regl1109
        movb #0x29,a3@+         | ')'
        rts
cmp9:   cmpb #9,d2
        bnes cmp10
        movw d1,d3
        andw #0x01c0,d3         | get op-mode
        cmpw #0x01c0,d3
        beqs dosuba
        cmpw #0x00c0,d3
        beqs dosuba
        btst #8,d1
        beqs dosub
        movw d1,d3
        andw #0x0030,d3
        bnes dosub
        lea pc@(subxmsg),a0
        bsr dmsg
        bsr prsize
        bra dpredec             | Print out either Dn,Dn or predecrement addr:
dosub:  lea pc@(submsg),a0
        bra opmode
dosuba: lea pc@(submsg),a0
        bsr dmsg
        btst #8,d1
        bnes subalng
        bsr dotw
        movb #1,a6@(127)        | set size flag to word
        bras subaea
subalng: bsr dotl
        movb #2,a6@(127)        | set size flag to long
subaea: movb #0x20,a3@+         | ' '
        bsr prea
comaA:  movb #0x2c,a3@+         | ','
        bra a11109
cmp10:  cmpb #10,d2
```

```
                beq badop
                cmpb #11,d2
                bne cmp12
                movw d1,d3
                andw #0x01c0,d3
                cmpw #0x00c0,d3
                beqs docmpa
                cmpw #0x01c0,d3
                beqs docmpa
                btst #8,d1
                bnes chkcmpm
                lea pc@(cmpmsg),a0
                bsr dmsg
                bsr prsizea
                bra comaD
docmpa:         lea pc@(cmpmsg),a0
                bsr dmsg
                btst #8,d1
                bnes cmpalng
                bsr dotw
                movb #1,a6@(127)        | set size flag to word
                bras cmpaea
cmpalng:        bsr dotl
                movb #2,a6@(127)        | set size flag to long
cmpaea:         movb #0x20,a3@+         | ' '
                bsr prea
                bra comaA
chkcmpm:        movw d1,d3
                andw #0x0038,d3
                cmpw #0x0008,d3
                bnes doeor
                lea pc@(cmpmsg),a0
                bsr dmsg
                bsr prsize
                movb #0x20,a3@+         | ' '
                movb #0x28,a3@+         | '('
                bsr prAreg
                lea pc@(parplus),a0     | ')+,('
                bsr dmsg
                bsr a11109
                movb #0x29,a3@+         | ')'
                movb #0x2b,a3@+         | '+'
                rts
doeor:          lea pc@(eormsg),a0
                bsr dmsg
                bsr prsize
                movb #0x20,a3@+         | ' '
                bsr d11109
                bra commaea
cmp12:          cmpb #12,d2
                bne cmp13
                movw d1,d3
                andw #0x01c0,d3
                cmpw #0x00c0,d3
                beqs domulu
                cmpw #0x01c0,d3
                beqs domuls
                cmpw #0x0140,d3
                beqs chkexgad
                cmpw #0x0180,d3
                beq chkexgm
                cmpw #0x0100,d3
                bnes doand
                movw d1,d3
                andw #0x0030,d3
                bnes doand
```

```
        lea pc@(abcdmsg),a0
        bsr dmsg
        bra dpredec
doand:  lea pc@(andmsg),a0
        bra opmode
domulu: lea pc@(mulumsg),a0
        bra msgeadn
domuls: lea pc@(mulsmsg),a0
        bra msgeadn
chkexgad: movw d1,d3
        andw #0x0038,d3
        bnes chkexga
        lea pc@(exgmsg),a0
        bsr dmsg
        bsr d11109
commaD: movb #0x2c,a3@+         | ','
        bra dreg
chkexga: cmpw #0x0008,d3
        bnes doand
        lea pc@(exgmsg),a0
        bsr dmsg
        bsr a11109
commaA: movb #0x2c,a3@+         | ','
        bra prAreg
chkexgm: movw d1,d3
        andw #0x0038,d3
        cmpw #0x0008,d3
        bnes doand
        lea pc@(exgmsg),a0
        bsr dmsg
        bsr d11109
        bras commaA
cmp13:  cmpb #13,d2
        bne cmp14
        movw d1,d3
        andw #0x01c0,d3
        cmpw #0x00c0,d3
        beqs addaddr
        cmpw #0x01c0,d3
        beqs addaddr
        btst #8,d1
        bnes chkaddx
doadd:  lea pc@(addmsg),a0
        bra opmode
addaddr: lea pc@(addmsg),a0
        bsr dmsg
        btst #8,d1
        bnes addr1
        bsr dotw
        movb #1,a6@(127)        | set size flag to word
        bras addaea
addr1:  bsr dotl
        movb #2,a6@(127)        | set size flag to long
addaea: movb #0x20,a3@+         | ' '
        bsr prea
        bra comaA
chkaddx: movw d1,d3
        andw #0x0030,d3
        bnes doadd
        movw d1,d3
        andw #0x00c0,d3
        cmpw #0x00c0,d3
        beq badop
        lea pc@(addxmsg),a0
        bsr dmsg
        bsr prsize
```

```
        bra dpredec
cmp14:  cmpb #14,d2
        bne badop
        movw d1,d3
        andw #0x00c0,d3
        cmpw #0x00c0,d3
        beq memshft
        movw d1,d3
        andw #0x0018,d3
        bnes rtype1
        lea pc@(asmsg),a0
shftmsg: bsr dmsg
        btst #8,d1
        bnes left
        movb #0x52,a3@+         | 'R'
        bras shftsize
left:   movb #0x4c,a3@+         | 'L'
shftsize: bsr prsize
        btst #5,d1
        bnes datshft
        bsr imm1to8
        bra commaD
datshft: movb #0x20,a3@+        | ' '
        bsr d11109
        bra commaD
rtype1: cmpw #0x0008,d3
        bnes rtype2
        lea pc@(lsmsg),a0
        bras shftmsg
rtype2: cmpw #0x0010,d3
        bnes rtype3
        lea pc@(roxmsg),a0
        bras shftmsg
rtype3: lea pc@(romsg),a0
        bras shftmsg
memshft: movw d1,d3
        andw #0x0600,d3
        bnes mtype1
        lea pc@(asmsg),a0
memmsg: bsr dmsg
        btst #8,d1
        bnes memleft
        movb #0x52,a3@+         | 'R'
        bras shiftea
memleft: movb #0x4c,a3@+        | 'L'
shiftea: bsr dotw
        movb #0x20,a3@+ | ' '
        bsr prea
        rts
mtype1: cmpw #0x0200,d3
        bnes mtype2
        lea pc@(lsmsg),a0
        bras memmsg
mtype2: cmpw #0x0400,d3
        bnes mtype3
        lea pc@(roxmsg),a0
        bras memmsg
mtype3: lea pc@(romsg),a0
        bras memmsg
badop:  lea pc@(questmsg),a0
        bsr dmsg
        rts
dmsg:   tstb a0@
        beqs dmsgdone
        movb a0@+,a3@+
        bras dmsg
```

```
dmsgdone: rts
prnspc: tstw d0
        beqs prndon
prnloop: movb #0x20,a1@+
        subqw #1,d0
        bnes prnloop
prndon: rts
prlongd0: swap d0
        bsrs prworddO
        swap d0
        bsrs prworddO
        rts
prworddO: movw d0,a7@-
        lsrw #8,d0
        bsrs prbyted0
        movw a7@+,d0
        bsrs prbyted0
        rts
prbyted0: movw d0,a7@-
        lsrb #4,d0
        bsr itoa
        movb d0,a1@+
        movw a7@,d0
        bsr itoa
        movb d0,a1@+
        movw a7@+,d0            | restore d0
        rts
prea:   movw d1,d2              | get back original to check other bits
        andw #0x0038,d2         | mask to read "mode"
        bnes mode1
        bra dreg
mode1:  cmpw #0x0008,d2         | check for mode = 1
        bnes mode2
        bra prAreg
mode2:  cmpw #0x0010,d2         | check for mode = 2
        bnes mode3
parApar: movb #0x28,a3@+        | '('
        bsr prAreg
        movb #0x29,a3@+         | ')'
        rts
mode3:  cmpw #0x0018,d2         | check for mode = 3
        bnes mode4
        bsrs parApar
        movb #0x2b,a3@+         | '+'
        rts
mode4:  cmpw #0x0020,d2
        bnes mode5
        movb #0x2d,a3@+         | '-'
        bras parApar
mode5:  cmpw #0x0028,d2
        bnes mode6
dispad: movw a2@+,d0
        bsr prworddO            | print word used in bytes field
        movb #0x20,a1@+         | ' '
        movb #0x24,a3@+         | '$'
        exg a1,a3               | ? should print '-' for neg. offsets ?
        bsr prworddO
        exg a1,a3
        bras parApar
mode6:  cmpw #0x0030,d2
        bnes mode7
        movw a2@+,d0
        bsr prworddO
        movb #0x20,a1@+         | ' '
        movb #0x24,a3@+         | '$'
        exg a1,a3
```

```
        bsr prbyted0            | print displacement
        exg a1,a3
        movb #0x28,a3@+         | '('
        bsr prAreg
prindex: movb #0x2c,a3@+        | ','
| d0 should still be intact, look at other bits
        btst #15,d0
        bnes adrindex           | Address register used as index if bit 15 se
        movb #0x44,a3@+         | 'D'ata register if zero
        bras saved06
adrindex: movb #0x41,a3@+       | 'A'
saved06: movw d0,d3             | save copy of word in d3
        rolw #4,d0
        andw #0x0007,d0         | mask off register bits
        addb #0x30,d0
        movb d0,a3@+            | print register number used for index
        btst #11,d3
        bnes lngindex           | size of index register used is long if 1
        bsr dotw
        bras clospar6
lngindex: bsr dotl
clospar6: movb #0x29,a3@+       | ')'
        rts
mode7:  movw d1,d2              | get back original opcode word
        andw #0x0007,d2         | get 'register number' field
        bnes mode71
        movw a2@+,d0            | get 16 bit address
        bsr prworddO
        movb #0x20,a1@+         | ' '
        movb #0x24,a3@+         | '$'
        exg a1,a3
        bsr prworddO
        exg a1,a3
        rts
mode71: cmpw #0x0001,d2         | reg = 1 ?
        bnes mode72
        movw a2@+,d0            | get high part of 32 bit address
        bsr prworddO
        movb #0x20,a1@+         | ' '
        swap d0                 | save half address in upper part of d0
        movw a2@+,d0
        bsr prworddO
        movb #0x20,a1@+         | ' '
        movb #0x24,a3@+         | '$'
        exg a1,a3
        bsr prlongd0
        exg a1,a3
        rts
mode72: cmpw #0x0002,d2         | reg = 2 ?
        bnes mode73
        movb #0x5b,a3@+         | '['
        movb #0x24,a3@+         | '$'
        movw a2@,d0
        movb #0x24,a3@+         | '$'
        movw a2@,d0
        extl d0
        addl a2,d0
        exg a1,a3
        bsr prlongd0
        exg a1,a3
        movb #0x5d,a3@+         | ']'
        movw a2@+,d0            | get 16 bit displacement
        bsr prworddO
        movb #0x20,a1@+         | ' '
        rts
mode73: cmpw #0x0003,d2         | reg = 3 ?
        bnes mode74
        movw a2@+,d0
```

```
        bsr prworddO
        movb #0x20,a1@+         | ' '
        movb #0x24,a3@+         | '$'
        exg a1,a3
        bsr prbytedO
        exg a1,a3
        lea pc@(pcmsg),aO
        bsr dmsg
        bra prindex
mode74: cmpw #0x0004,d2         | reg = 4 ?
        bnes badmode7
        movb #0x23,a3@+         | '#'
        movb #0x24,a3@+         | '$'
        movw a2@+,dO
        bsr prworddO
        movb #0x20,a1@+         | ' '
        exg a1,a3
        movb a6@(127),d3        | get size flag
        bnes seew
        bsr prbytedO
restala3: exg a1,a3
        rts
seew:   cmpb #1,d3
        bnes longimm
        bsr prworddO
        bras restala3
longimm: swap dO
        movw a2@+,dO
        exg a1,a3               | restore to originals for printing
        bsr prworddO
        movb #0x20,a1@+         | ' '
        exg a1,a3
        bsr prlongdO
        bras restala3
badmode7: lea pc@(questmsg),aO
        bsr dmsg
        rts
prmvdst: rorw #3,d1
        movw d1,d2
        andw #0x0038,d2
        rorw #6,d1
        andw #0x0007,d1
        orw d2,d1
        bra prea                | now that d1 has been fudged to contain
| the mode and register in correct place go print just like source EA
prregnum: movb d1,d2            | get back original to read reg number
        andb #0x07,d2
        addb #0x30,d2
        movb d2,a3@+
        rts
dotb:   movb #0x2e,a3@+         | '.'
        movb #0x42,a3@+         | 'B'
        rts
dotw:   movb #0x2e,a3@+         | '.'
        movb #0x57,a3@+         | 'W'
        rts
dotl:   movb #0x2e,a3@+         | '.'
        movb #0x4c,a3@+         | 'L'
        rts
dots:   movb #0x2e,a3@+         | '.'
        movb #0x53,a3@+         | 'S'
        rts
prsize: movw d1,d3
        andw #0x00c0,d3         | get bits 7,6 for size info
        bnes sizewd
        bsrs dotb
```

```
        movb #0,a6@(127)        | set size flag to byte
        rts
sizewd: cmpw #0x0040,d3
        bnes sizelng
        bsrs dotw
        movb #1,a6@(127)        | set size flag to word
        rts
sizelng: bsrs dotl
        movb #2,a6@(127)        | set size flag to long
        rts
primm:  movb a6@(127),d2
        bnes immwd
        bsr prbytimm
        rts
immwd:  cmpb #1,d2
        bnes immlng
        bsr prwdimm
        rts
immlng: movw a2@+,d0
        bsr prworddO
        movb #0x20,a1@+         | ' '
        swap d0
        movw a2@+,d0
        bsr prworddO
        movb #0x20,a1@+         | ' '
        bsrs spnumdol
        exg a1,a3
        bsr prlongd0
        exg a1,a3
        movb #0x2c,a3@+         | ','
        rts
spnumdol: movb #0x20,a3@+       | ' '
        movb #0x23,a3@+         | '#'
        movb #0x24,a3@+         | '$'
        rts
prsizea: bsr dmsg
        bsr prsize
        movb #0x20,a3@+         | ' '
        bsr prea
        rts
preglist: movw a2@+,d0
        bsr prworddO
        movb #0x20,a1@+
regsonly: movw d1,d3
        andw #0x0038,d3
        cmpw #0x0020,d3
        beqs predecmv
        tstw d0                 | see if register list is empty
        bnes yesregs
        lea pc@(noregmsg),a0
        bsr dmsg
        rts
yesregs: movb #8,d2             | bit counter
        movb #0,d3              | flag, set when first reg printed
addrloop: lslw #1,d0
        bccs nextadr
        tstb d3
        beqs printA
        movb #0x2f,a3@+         | '/'
printA: movb #0x41,a3@+         | 'A'
        addb #0x2f,d2
        movb d2,a3@+
        subb #0x2f,d2
        addqb #1,d3
nextadr: subqb #1,d2
        bnes addrloop
```

```
        movb #8,d2
dataloop: lslw #1,d0
        bccs nextdata
        tstb d3
        beqs printD
        movb #0x2f,a3@+         | '/'
printD: movb #0x44,a3@+         | 'D'
        addb #0x2f,d2
        movb d2,a3@+
        subb #0x2f,d2
        addqb #1,d3
nextdata: subqb #1,d2
        bnes dataloop
        rts
predecmv:
        tstw d0                 | see if register list is empty
        bnes realreg
        lea pc@(noregmsg),a0
        bsr dmsg
        rts
realreg: movb #8,d2             | bit counter
        movb #0,d3              | flag, set when first reg printed
addrlup: lsrw #1,d0
        bccs nxtadr
        tstb d3
        beqs prntA
        movb #0x2f,a3@+         | '/'
prntA:  movb #0x41,a3@+         | 'A'
        addb #0x2f,d2
        movb d2,a3@+
        subb #0x2f,d2
        addqb #1,d3
nxtadr: subqb #1,d2
        bnes addrlup
        movb #8,d2
datalup: lsrw #1,d0
        bccs nxtdata
        tstb d3
        beqs prntD
        movb #0x2f,a3@+         | '/'
prntD: movb #0x44,a3@+          | 'D'
        addb #0x2f,d2
        movb d2,a3@+
        subb #0x2f,d2
        addqb #1,d3
nxtdata: subqb #1,d2
        bnes datalup
        rts
prcond: movw d1,d3
        andw #0x0f00,d3         | get condition code bits
        bnes cond1
        movb #0x54,a3@+         | 'T'
        rts
cond1:  cmpw #0x0100,d3
        bnes cond2
        movb #0x46,a3@+         | 'F'
        rts
cond2:  cmpw #0x0200,d3
        bnes cond3
        lea pc@(himsg),a0
        bra condmsg
cond3:  cmpw #0x0300,d3
        bnes cond4
        lea pc@(lsmsg),a0
        bra condmsg
cond4:  cmpw #0x0400,d3
```

```
        bnes cond5
        lea pc@(hsmsg),a0
        bras condmsg
cond5:  cmpw #0x0500,d3
        bnes cond6
        lea pc@(lomsg),a0
        bras condmsg
cond6:  cmpw #0x0600,d3
        bnes cond7
        lea pc@(nemsg),a0
        bras condmsg
cond7:  cmpw #0x0700,d3
        bnes cond8
        lea pc@(eqmsg),a0
        bras condmsg
cond8:  cmpw #0x0800,d3
        bnes cond9
        lea pc@(vcmsg),a0
        bras condmsg
cond9:  cmpw #0x0900,d3
        bnes cond10
        lea pc@(vsmsg),a0
        bras condmsg
cond10: cmpw #0x0a00,d3
        bnes cond11
        lea pc@(plmsg),a0
        bras condmsg
cond11: cmpw #0x0b00,d3
        bnes cond12
        lea pc@(mimsg),a0
        bras condmsg
cond12: cmpw #0x0c00,d3
        bnes cond13
        lea pc@(gemsg),a0
        bras condmsg
cond13: cmpw #0x0d00,d3
        bnes cond14
        lea pc@(ltmsg),a0
        bras condmsg
cond14: cmpw #0x0e00,d3
        bnes cond15
        lea pc@(gtmsg),a0
        bras condmsg
cond15: lea pc@(lemsg),a0
condmsg: bsr dmsg
        rts
hcom:   movb #0xff,d0
        bras savflg
ucom:   clrb d0
savflg: movw d0,sp@-
        movb #0xff,rawflag      | Stop Cntl-S/Q from affecting local I/O
comlp:  bsr chkc
        beqs tryucm
        cmpb #0x01,d0
        beqs uret
        tstb sp@(1)
        beqs noecho
        jsr a5@
noecho: bsr putcm
tryucm: bsr chkcm
        beqs comlp
        jsr a5@
        bras comlp
uret:   addql #2,sp
        clrb rawflag            | Restore local pause capability
        rts
```

```
help:   lea pc@(helpmsg),a0
        bsr msg
        rts


|*** initialize duart1 ***
|*** write to odd address so to put byte in correct position
|*** -- mrk

initport: movb #0x02,0x200001   | Set Mode Register 1A
        movb #0x07,0x200001     | Set Mode Register 2A
        movb #0xbb,0x200003     | Set CSRA for 9600 baud for terminal
        movb #0x80,0x200009     | Set Aux. Control Register (ACR)
        movb #0x33,0x20000b     | Set Interrupt Mask(IMR) for Channel A+B XMT
        movb #0xff,0x20000d     | Set Counter Upper Register
        movb #0xff,0x20000f     | Set Counter Lower Register
        movb #0x02,0x200011     | Set Mode Register 1B
        movb #0x07,0x200011     | Set Mode Register 2B
        movb #0xbb,0x200013     | Set CSRB for 9600 baud for modem/download
        movb #0x40,0x200019     | Set interrupt vector -> $100 in memory
        movb #0x00,0x20001b     | Set Output Control Register (OPCR)
        movb #0x05,0x200005     | Set Command Register A to enable XMIT and R
        movb #0x05,0x200015     | Set Command Register B to enable XMIT and R
        rts                     | remove when duart2 installed

|*** initialize duart2 ***

        movb #0x02,0x202001     | Set Mode Register 1A
        movb #0x07,0x202001     | Set Mode Register 2A
        movb #0xbb,0x202003     | Set CSRA for 9600 baud for terminal
        movb #0x80,0x202009     | Set Aux. Control Register (ACR)
        movb #0x33,0x20200b     | Set Interrupt Mask(IMR) for Channel A+B XMT
        movb #0xff,0x20200d     | Set Counter Upper Register
        movb #0xff,0x20200f     | Set Counter Lower Register
        movb #0x02,0x202011     | Set Mode Register 1B
        movb #0x07,0x202011     | Set Mode Register 2B
        movb #0xbb,0x202013     | Set CSRB for 9600 baud for modem/download
        movb #0x44,0x202019     | Set interrupt vector -> $104x in memory
        movb #0x00,0x20201b     | Set Output Control Register
        movb #0x05,0x202005     | Set Command Register A to enable XMIT and R
        movb #0x05,0x202015     | Set Command Register B to enable XMIT and R
        rts                     | return

|*****************************************************************************
|*** TERMINAL 1 GETC
|*****************************************************************************

getc:   bsrs chkc
        beqs getc
        rts
chkc:   moveml #0x6080,sp@-     | save a0,d1 on stack *CHANGED
        movw sr,d1              | save current status word in d2 *ADDED
        movw d1,d2              |save only current priority mask *ADDED
        andw #0x0700,d2
        andw #0xf8ff,d1         | *CHANGED
        orw #0x0700,d1          | Disable all maskable interrupts *CHANGED
        movw d1,sr
        movl TRCVTAIL,a0
        cmpl TRCVHEAD,a0
        bnes getok
endgetc: movw sr,d1
        andw #0xf8ff,d1         | put old priority level back *ADDED
        orw d2,d1               | Re-enable interrupts
        movw d1,sr              | restore a0,d1,d2 from stack *CHANGED
        moveml sp@+,#0x0106
        rts
```

```
getok:   movb a0@+,d0
         cmpl #TXMTBUF,a0
         bnes getcsta0
         movl #TRCVBUF,a0
getcsta0: movl a0,TRCVTAIL
         bras endgetc


|*****************************************************************************
|*** MODEM GETC
|*****************************************************************************

getcm:   bsrs chkcm
         beqs getcm
         rts
chkcm:   moveml #0x6080,sp@-        | save a0,d1,d2 on stack *CHANGED
         movw sr,d1
         movw d1,d2                 |save current status word in d2 *ADDED
         andw #0x0700,d2            |save only current priority mask *ADDED
         andw #0xf8ff,d1
         orw #0x0700,d1             |disable all maskable interrupts *CHANGED
         movw d1,sr
         movl MRCVTAIL,a0
         cmpl MRCVHEAD,a0
         bnes getmok
endgetcm: movw sr,d1
         andw #0xf8ff,d1
         orw d2,d1                  |put old priority_level back *ADDED
         movw d1,sr
         moveml sp@+,#0x0106        |restore a0,d1,d2 *CHANGED
         rts
getmok:  movb a0@+,d0
         cmpl #MXMTBUF,a0
         bnes getcmsta
         movl #MRCVBUF,a0
getcmsta: movl a0,MRCVTAIL
         bras endgetcm

|*****************************************************************************
|*** TERMINAL 1 PUTC
|*****************************************************************************

putc:    moveml #0xe0c0,sp@-        | save d0,d1,d2,a0,a1 on stack *CHANGED
putcwt:  movw sr,d1
         movw d1,d2                 | save old status regester in d2 *ADDED
         andw #0x0700,d2            | save only the old priority level in d2 *ADD
         andw #0xf8ff,d1
         orw #0x0700,d1             | Turn off all maskable interrupts *CHANGED
         movw d1,sr
         movl TXMTHEAD,a0
         lea a0@(1),a1
         cmpl #MRCVBUF,a1
         bcss putccmp
         movl #TXMTBUF,a1
putccmp: cmpl TXMTTAIL,a0
         beqs newtxmt
         cmpl TXMTTAIL,a1
         bnes dosto
         movw sr,d1
         andw #0xf8ff,d1
         orw d2,d1                  | put back the old priority level *ADDED
         movw d1,sr                 | enable interrupts again
         bras putcwt                | loop till there is room in queue
dosto:   movb d0,a0@
         movl a1,TXMTHEAD
endputc: movw sr,d1
```

```
        andw #0xf8ff,d1
        orw d2,d1                   | put back the old priority level *ADDED
        movw d1,sr                  | enable interrupts again
        moveml sp@+,#0x0307         | restore a1,a0,d2,d1,d0 from stack *CHANGED
        rts
newtxmt: movb d0,a0@
        movl a1,TXMTHEAD
        movb #0x04,0x200005         | Enable Channel A (terminal) transmitter
        bras endputc

|*****************************************************************************
|*** MODEM PUTC
|*****************************************************************************

putcm:  moveml #0xe0c0,sp@-         | save d0,d1,d2,a0,a1 on stack *CHANGED
putcmwt: movw sr,d1
        andw #0xf8ff,d1
        movw d1,d2                  | save old status register in d2 *ADDED
        andw #0x0700,d2             | save only the old priority level in d2 *ADDI
        orw #0x0700,d1              | Turn off all maskable interrupts *CHANGED
        movw d1,sr
        movl MXMTHEAD,a0
        lea a0@(1),a1
        cmpl #KEYBUF,a1
        bcss putcmcmp
        movl #MXMTBUF,a1
putcmcmp: cmpl MXMTTAIL,a0
        beqs newmxmt
        cmpl MXMTTAIL,a1
        bnes dostom
        movw sr,d1
        andw #0xf8ff,d1
        orw d2,d1                   | put back the old priority level *ADDED
        movw d1,sr                  | enable interrupts again
        bras putcmwt
dostom: movb d0,a0@
        movl a1,MXMTHEAD
endputcm: movw sr,d1
        andw #0xf8ff,d1
        orw d2,d1                   | put back the old priority level *ADDED
        movw d1,sr                  | enable interrrupts again
        moveml sp@+,#0x0307         | restore a1,a0,d2,d1,d0 from stack *CHANGED
        rts
newmxmt: movb d0,a0@
        movl a1,MXMTHEAD
        movb #0x04,0x200015         | Enable Channel B (modem) transmitter
        bras endputcm

|*****************************************************************************
|*****************************************************************************
|*****************************************************************************

inchar: clrw d0                     | Make sure high byte is cleared
        tstb editflg                | See if direct or line input
        beqs getcdir                | Go get directly if off
        tstb lineflg                | Do we already have a buffer full ?
        bnes bufchr                 | Yes, go get from buffer
        bsrs ugetbuf                | No, go get a buffer full
        movb #0xff,lineflg          | Set lineflg since we now have buffe:
bufchr: movl a0,sp@-                | Save current a0 register
        movl BUFPTR,a0              | Point a0 to buffer
        movb a0@+,d0                | get byte from buffer
        movl a0,BUFPTR              | put inc'ed pointer back
        cmpb #0x0a,d0               | Newline char ?
        bnes retchar                | No, just return it
```

```
        clrb lineflg                    | Yes, show buffer is empty now
retchar: movl sp@+,a0                   | Restore a0 register
        rts
getcdir: movl a0,sp@-                   | Save current a0 register
        movl getchr,a0                  | Get address of getchr routine in a0
        clrw d0                         | Make sure high byte of d0 is clear
        jsr a0@                         | Get a character from the monitor
        cmpb eolchar,d0                 | Check for end-of-line character
        bnes gchar2                     | No, don't convert char.
        movb #0x0a,d0                   | Yes, convert to '\n'
gchar2: bsr uputc                       | Call "UNIX putchr" routine
        movl sp@+,a0                    | Restore old a0 value
        rts
ugetbuf: moveml #0x40e0,sp@-            | Save d1,a0,a1,a2 on stack
        movl #buffer,a1                 | Point a1 to keyboard buffer
        movl a1,BUFPTR                  | Also point BUFPTR to buffer
        movb #0xff,d1                   | Maximum # of chars to get in buffer
        movl getchr,a0                  | Get address of getchr routine in a0
        movl putchr,a2                  | Get address of putchr routine in a2
uchrloop: jsr a0@                       | Get a char. from the monitor
        cmpb #0x08,d0                   | Check for backspace
        beqs backspc
        cmpb killchar,d0                | Check for line kill character
        bnes chkeol                     | No, go check for end-of-line char.
erase:  cmpb #0xff,d1                   | Any room to backspace on line ?
        beqs uchrloop                   | No, just go get another char.
        movb #0x08,d0                   | Load backspace character
        jsr a2@                         | Print character using monitor
        addb #1,d1                      | Room for one more char in buffer no
        subl #1,a1                      | Move back buffer pointer
        bras erase                      | And loop till done
chkeol: cmpb eolchar,d0                 | End-of-line character ?
        bnes seelf                      | No, check for linefeed
        movb #0x0a,d0                   | Yes, convert to '\n'
        bras sendlf
seelf:  cmpb #0x0a,d0                   | Linefeed character, always EOL
        bnes chkbslsh                   | No, go check for backslash
sendlf: bsrs uputc                      | Call "UNIX putchr" routine
        movb d0,a1@                     | Put char in buffer
uexit:  moveml sp@+,#0x0702             | Restore d1,a0,a1,a2
        rts
chkbslsh: bsrs uputc                    | Echo initial character
        cmpb #0x5c,d0                   | Check for '\'
        bnes ustochr                    | No, just store character in buffer
        jsr a0@                         | Yes, go get another character
        bsr uputc                       | Echo new char just gotten
ustochr: movb d0,a1@+                   | Store char in buffer, inc pointer
        subb #1,d1                      | Room for one less character
        bnes uchrloop                   | If non-zero then get more chars
        bras uexit                      | Yes, buffer full, return
backspc: cmpb #0xff,d1                  | Room to backspace on line ?
        beqs uchrloop                   | No, just ignore backspace
        jsr a2@                         | Yes, print backspace using monitor
        addb #1,d1                      | Room for one more char in buffer
        subl #1,a1                      | Move buffer pointer back one
        bras uchrloop                   | Go get more characters

|## This routine prints a character, doing UNIX style newline and
|## carriage return translations on output
|## Now setup arguments to pass to outchar routine which does real work

uputc:  movl #putchr,sp@-               | Pass address of putchr to outchar
        movw d0,sp@-                    | Pass char to print to outchar
        bsrs outchar                    | Print char, let outchar worry about
                                        | NL and CR translations
        movw sp@+,d0                    | Restore character to d0
```

```
        addl #4,sp                      | Get address off stack
        rts
outchar:
        movb sp@(5),d0                  | Get char to print passed on stack
        movl a0,sp@-                    | Save current a0 register
        cmpb #0x0a,d0                   | Is it a newline char ?
        bnes put1                       | No, handle normally
        movb #0x0d,d0                   | Yes, load a <CR> to print first
        movl sp@(10),a0                 | Get I/O address passed on stack
        movl a0@,a0                     | Get pointer from pointer
        jsr a0@                         | Print out character
        movb #0x0a,d0                   | Reload the linefeed char. to print
put1:   movl sp@(10),a0                 | Get I/O address passed on stack
        movl a0@,a0                     | Get pointer from pointer
        jsr a0@                         | Print out character
        movl sp@+,a0                    | Restore a0
        rts

|*****************************************************************************
|*** INITIALIZE DUART1
|*****************************************************************************

duartlint: moveml #0x80c0,sp@-  | Save d0,a0,a1 on stack
        movb 0x20000b,d0                | read interrupt status register
        btst #1,d0                      | Terminal receiver interrupt?
        bnes trmrcv
        btst #0,d0                      | Terminal transmitter interrupt?
        bne trmxmit
        btst #5,d0                      | Modem receiver interrupt?
        bne mdmrcv
        btst #4,d0                      | Modem transmitter interrupt?
        bne mdmxmit
| It should never reach the following code unless DUART mis-programmed!
        lea pc@(badintms),a0
        bsr crlfmsg
        moveml sp@+,#0x0301             | restore a1,a0,d0
        rte
trmrcv: movl TRCVHEAD,a0
        lea a0@(1),a1
        cmpl #TXMTBUF,a1
        bnes docmpl
        movl #TRCVBUF,a1
docmpl: cmpl TRCVTAIL,a1
        beqs trmiend                    | Buffer overflow
        tstb tslshflg
        beqs seeslsh
        clrb tslshflg
        movb 0x200007,d0                | read byte from UART
trmistor: movb d0,a0@                   | store byte in buffer
        movl a1,TRCVHEAD
tend:   moveml sp@+,#0x0301             | restore a1,a0,d0
        rte
trmiend: movb 0x200007,d0               | read byte to clear interrupt
        bras tend
seeslsh: movb 0x200007,d0               | read byte from UART
        cmpb #0x5c,d0                   | check for `\`
        bnes seecntl
        movb #0xff,tslshflg
        bras trmistor
seecntl: cmpb intrchar,d0               | was it the interrupt character ?
        bnes seepause
        orw #0x0700,sr                  | set interrupt level to 7 to turn all off
        lea pc@(mainst),a0
        movl a0,sp@(14)                 | Fudge return address
        moveml sp@+,#0x0301             | restore a1,a0,d0 to get off stack
        rte                             | Now return to fudged (mainst) address
```

```
seepause: tstb rawflag          | Are we communicating with a remoter compute
          bnes trmistor         | Yes, then don't process Cntl-S or Cntl-Q!
          cmpb #0x13,d0         | was it a control-S ?
          bnes seecntlq
          movb #0xff,pausflg    | yes, set flag to pause output
          bras tend             | Don't store Cntl-S in input buffer!
seecntlq: cmpb #0x11,d0         | was it a control-Q ?
          bnes trmistor
          tstb pausflg
          beqs tend             | If pausflg wasn't set then ignore Cntl-Q
          clrb pausflg          | Clear flag to allow output to resume
          moveml #0x80c0,sp@-   | Save d0,a0,a1 on stack
          movl TXMTTAIL,a0
          movb a0@+,0x200007    | write data to UART
| To be absolutely safe you might want to check the status of the UART to
| see if Xmitter is empty, but since no one should be able to type a Cnlt-Q
| so fast after a Cntl-S that the buffer is not empty I won't worry about it.
          cmpl #MRCVBUF,a0
          bcss newtxptr
          movl #TXMTBUF,a0
newtxptr: movl a0,TXMTTAIL
          movb #0x04,0x200005   | Enable Channel A (terminal) transmitter
          moveml sp@+,#0x0301   | restore a1,a0,d0 from stack
          bra tend              | Don't store Cntl-Q in input buffer!
trmxmit: movl TXMTTAIL,a0
          cmpl TXMTHEAD,a0
          beqs notrmxmt
          tstb pausflg          | Are we trying to pause the output stream?
          bnes notrmxmt         | Yes, stop output till flag gets cleared
          movb a0@+,0x200007    | write data to UART
          cmpl #MRCVBUF,a0
          bcss savtxptr
          movl #TXMTBUF,a0
savtxptr: movl a0,TXMTTAIL
          moveml sp@+,#0x0301   | restore a1,a0,d0
          rte
notrmxmt: movb #0x08,0x200005   | Disable Channel A transmitter
          moveml sp@+,#0x0301   | restore a1,a0,d0
          rte
mdmrcv:  movl MRCVHEAD,a0
          lea a0@(1),a1
          cmpl #MXMTBUF,a1
          bnes docmp2
          movl #MRCVBUF,a1
docmp2:  cmpl MRCVTAIL,a1
          beqs mdmiend          | Buffer overflow
          movb 0x200017,a0@     | read byte from UART and store in buffer
          movl a1,MRCVHEAD
mend:     moveml sp@+,#0x0301   | restore a1,a0,d0
          rte
mdmiend: movb 0x200017,d0       | Read byte to clear interrupt
          bras mend
mdmxmit: movl MXMTTAIL,a0
          cmpl MXMTHEAD,a0
          beqs nomdmxmt
          movb a0@+,0x200017    | write data to UART
          cmpl #KEYBUF,a0
          bcss savmxptr
          movl #MXMTBUF,a0
savmxptr: movl a0,MXMTTAIL
          moveml sp@+,#0x0301   | restore a1,a0,d0
          rte
nomdmxmt: movb #0x08,0x200015   | Disable Channel B transmitter
          moveml sp@+,#0x0301   | restore a1,a0,d0
          rte
```

```
|*****************************************************************************
|*** INITIALIZE DUART2
|*****************************************************************************

|*****************************************************************************
|*****************************************************************************

buserr: lea pc@(busmsg),a0
busaderr: lea pc@(putc),a5
        movl a5,putchr
        bsr crlfmsg
        btst #4,sp@(1)  | See if it was a read or write instruction *CHANGED
        beqs writmsg            | Zero for write, One for read
        lea pc@(readmsg),a0
        bsr msg
        bras prbadadr
writmsg: lea pc@(writemsg),a0
        bsr msg
prbadadr: movl sp@(2),a0
        bsr prlong
        lea pc@(procadmsg),a0   | *ADDED
        bsr crlfmsg             | *ADDED
        movb sp@(1),d0          | get byte containing function code *ADDED
        bsr prfc                | print out deciphered function code *ADDED
        lea pc@(opcodmsg),a0
        bsr crlfmsg
        movw sp@(6),d0          | Get instruction that caused error
        bsr prword              | Print it out in hex
        lea pc@(badpcmsg),a0
        bsr crlfmsg
        movl sp@(10),a0         | get program counter where error occurred
        subl #2,a0      | correct PC back to about start of instruction *ADDEI
        bsr prlong
        lea pc@(fatalmsg),a0
        bsr crlfmsg
        orw #0x0700,sr          | set interrupt level to 7 to turn all off
        addl #10,sp             | Clean up exception info on stack
        lea pc@(mainst),a0
        movl a0,sp@             | Fudge return address
        rts                     | Now return to fudged (mainst) address
addrerr: lea pc@(addrmsg),a0
        bra busaderr            | *** changed bras to bra
illegal: lea pc@(illinmsg),a0
excepmsg: lea pc@(putc),a5
        movl a5,putchr
        bsr crlfmsg
        lea pc@(fatalmsg),a0
        bsr crlfmsg
        lea pc@(mainst),a0
        movl a0,sp@(2)
        rtr
divzero: lea pc@(zeromsg),a0
        bras excepmsg
uninit: moveml #0x80c0,sp@-     | Save d0,a0,a1 on stack
        lea pc@(uninitms),a0
        bsr crlfmsg
        moveml sp@+,#0x0301     | Restore a1,a0,d0
        rte
prfc:                           | print function code meaning *ADDED
        andb #0x07,d0           | mask to get only Function Code *ADDED
        cmpb #1,d0              | user data ? *ADDED
        bnes tryup              | *ADDED
        lea pc@(usedatmsg),a0   | *ADDED
        bras prfcmsg            | *ADDED
tryup:  cmpb #2,d0              | user program ?  *ADDED
        bnes trysupd            | *ADDED
```

```
        lea pc@(useprgmsg),a0   | *ADDED
        bras prfcmsg            | *ADDED
trysupd: cmpb #4,d0             | supervisor data ? *ADDED
        bnes trysupp            | *ADDED
        lea pc@(supdatmsg),a0   | *ADDED
        bras prfcmsg            | *ADDED
trysupp: cmpb #5,d0             | supervisor program ? *ADDED
        bnes tryintak           | ADDED
        lea pc@(supprgmsg),a0   | ADDED
        bras prfcmsg            | ADDED
tryintak: cmpb #7,d0            | interrupt acknowledge ? *ADDED
        bnes havbadfc           | *ADDED
        lea pc@(intackmsg),a0   | *ADDED
        bras prfcmsg            | *ADDED
havbadfc: lea pc@(badfcmsg),a0  | *ADDED
prfcmsg: bsr msg                | *ADDED
        rts
        .word 0
        .data
cmndtbl: .byte 0x45             | 'E'
        .byte 0
        .long exam-monref
        .byte 0x47              | 'G'
        .byte 0
        .long go-monref
        .byte 0x57              | 'W'
        .byte 0
        .long write-monref
        .byte 0x4c              | 'L'
        .byte 0
        .long load-monref
        .byte 0x4d              | 'M'
        .byte 0
        .long mov-monref
        .byte 0x44              | 'D'
        .byte 0
        .long disasmbl-monref
        .byte 0x55              | 'U'
        .byte 0
        .long ucom-monref
        .byte 0x3f              | '?'
        .byte 0
        .long help-monref
        .byte 0x48              | 'H'
        .byte 0
tblend: .long hcom-monref
sysmsg: .ascii "MC68000  Tutor 1.7.1"
        .byte 0x0d
        .byte 0x0a
        .ascii "August 27, 1986."
        .byte 0x0d
        .byte 0x0a
        .byte 0x0d
        .byte 0x0a
        .ascii "Type ? command for help"
        .byte 0x0d
        .byte 0x0a
        .byte 0
prmptmsg: .asciz "TUTOR==> "
ntfndmsg: .asciz "Command Not Found"
bargmsg: .asciz "Bad or wrong # of args"
error:  .asciz "What do you really mean?"
bdldmsg: .asciz "Bad load at "
chksmsg: .asciz "Checksum error at "
typemsg: .asciz "File type mismatch"
bdbytmsg: .asciz "Bad byte received at "
```

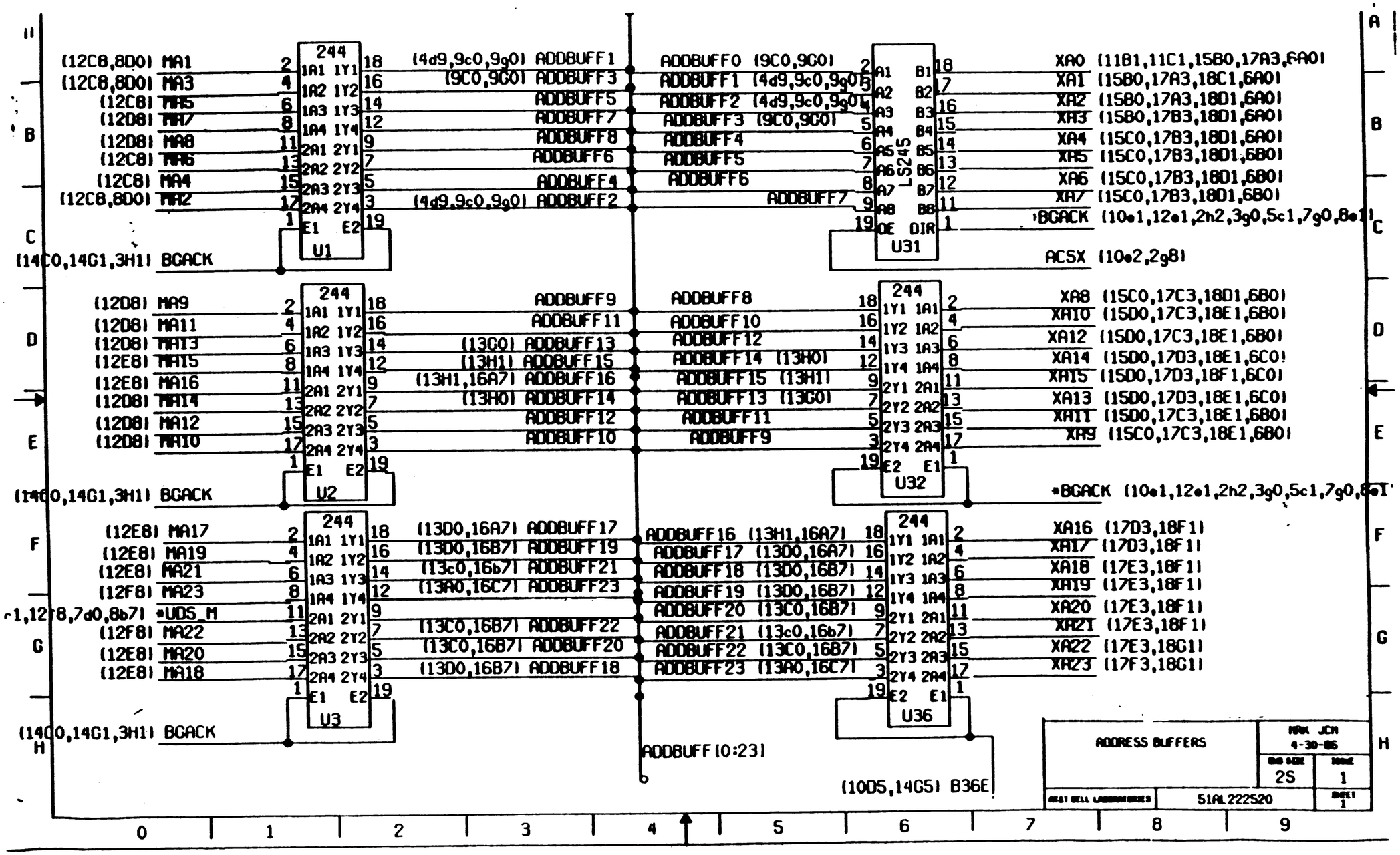
```
busmsg: .asciz "A BUS ERROR OCCURRED DURING A "
addrmsg: .asciz "AN ADDRESSING ERROR OCCURRED DURING A "
readmsg: .asciz "READ OF ADDRESS Ox"
writemsg: .asciz "WRITE TO ADDRESS Ox"
procadmsg: .asciz "PROCESSOR WAS ATTEMPTING TO "          | *ADDED
usedatmsg: .asciz "ACCESS USER DATA MEMORY "              | *ADDED
useprgmsg: .asciz "ACCESS USER PROGRAM MEMORY "           | *ADDED
supdatmsg: .asciz "ACCESS SUPERVISOR DATA MEMORY "        | *ADDED
supprgmsg: .asciz "ACESS SUPERVISOR PROGRAM MEMORY "      | *ADDED
intackmsg: .asciz "DO AN INTERRUPT ACKNOWLEDGE "          | *ADDED
badfcmsg: .asciz "ACCESS MEMORY WITH AN UNASSIGNED FUNCTION CODE !!"| *ADDED
opcodmsg: .asciz "INSTRUCTION OPCODE (hex) = "
badpcmsg: .asciz "PROGRAM COUNTER FOR ERRANT INSTRUCTION = Ox"
illinmsg: .asciz "ILLEGAL INSTRUCTION DETECTED"
zeromsg: .asciz "ATTEMPTED DIVIDE BY ZERO"
fatalmsg: .asciz "FATAL ERROR - RETURNING TO MONITOR"
badintms: .asciz "UNDECIPHERABLE INTERRUPT FROM DUART1 RECEIVED!"
uninitms: .asciz "UNINITIALIZED VECTOR INTERRUPT RECEIVED!"
questmsg: .asciz "????"
ormsg:   .asciz  "OR"
ccmsg:   .asciz  "CCR"
movmsg: .asciz "MOVE"
pcmsg:   .asciz "(PC"
srmsg:   .asciz "SR"
andmsg: .asciz "AND"
eormsg: .asciz "EOR"
movepmsg: .asciz "MOVEP"
tstmsg: .asciz "TST"
chgmsg: .asciz "CHG"
clrmsg: .asciz "CLR"
setmsg: .asciz "SET"
addmsg: .asciz "ADD"
submsg: .asciz "SUB"
cmpmsg: .asciz "CMP"
negmsg: .asciz "NEG"
negxmsg: .asciz "NEGX"
movfsr: .asciz "MOVE.W SR,"
notmsg: .asciz "NOT"
swapmsg: .asciz "SWAP"
extmsg: .asciz "EXT"
nbcdmsg: .asciz "NBCD"
peamsg: .asciz "PEA"
movemwms: .asciz "MOVEM.W"
movemlms: .asciz "MOVEM.L"
illmsg: .asciz "ILLEGAL"
tasmsg: .asciz "TAS"
resetmsg: .asciz "RESET"
trapmsg: .asciz "TRAP"
linkmsg: .asciz "LINK"
unlkmsg: .asciz "UNLK"
uspmsg: .asciz "USP"
nopmsg: .asciz "NOP"
stopmsg: .asciz "STOP"
rtemsg: .asciz "RTE"
rtsmsg: .asciz "RTS"
trapvmsg: .asciz "TRAPV"
rtrmsg: .asciz "RTR"
jsrmsg: .asciz "JSR"
jmpmsg: .asciz "JMP"
chkmsg: .asciz "CHK"
leamsg: .asciz "LEA"
himsg:   .asciz "HI"
lsmsg:   .asciz "LS"
hsmsg:   .asciz "HS(CC)"
lomsg:   .asciz "LO(CS)"
nemsg:   .asciz "NE"
```

```
eqmsg:   .asciz "EQ"
vcmsg:   .asciz "VC"
vsmsg:   .asciz "VS"
plmsg:   .asciz "PL"
mimsg:   .asciz "MI"
gemsg:   .asciz "GE"
ltmsg:   .asciz "LT"
gtmsg:   .asciz "GT"
lemsg:   .asciz "LE"
bramsg:  .asciz "BRA"
brnmsg:  .asciz "BRN"
bsrmsg:  .asciz "BSR"
moveqmsg:  .asciz "MOVEQ.L #$"
divumsg:  .asciz "DIVU"
divsmsg:  .asciz "DIVS"
sbcdmsg:  .asciz "SBCD.B"
negparA:  .asciz "-(A"
subxmsg:  .asciz "SUBX"
spcparA:  .asciz " (A"
parplus:  .asciz ")+,("
mulumsg:  .asciz "MULU"
mulsmsg:  .asciz "MULS"
abcdmsg:  .asciz "ABCD.B"
exgmsg:  .asciz "EXG "
addxmsg:  .asciz "ADDX"
asmsg:   .asciz "AS"
roxmsg:  .asciz "ROX"
romsg:   .asciz "RO"
noregmsg:  .asciz "No Registers"
helpmsg:  .ascii "The following commands are available:"
          .byte 0x0d
          .byte 0x0a
          .ascii "D hexaddress"
          .byte 0x0d
          .byte 0x0a
          .ascii "       Disassemble machine code to screen"
          .byte 0x0d
          .byte 0x0a
          .ascii "E hexaddress"
          .byte 0x0d
          .byte 0x0a
          .ascii "       Examine and change memory locations"
          .byte 0x0d
          .byte 0x0a
          .ascii "G hexaddress"
          .byte 0x0d
          .byte 0x0a
          .ascii "       Go execute machine code at the hex address specified"
          .byte 0x0d
          .byte 0x0a
          .ascii "H      Half-duplex communication with modem port"
          .byte 0x0d
          .byte 0x0a
          .ascii "L[X] [hexoffset]"
          .byte 0x0d
          .byte 0x0a
          .ascii "       Load - Download Intel hex format machine code with opti
          .byte 0x0d
          .byte 0x0a
          .ascii "                Use the X option to load extended address (32 bi
          .byte 0x0d
          .byte 0x0a
          .ascii "M start,end,destination  (hex addresses)"
          .byte 0x0d
          .byte 0x0a
          .ascii "       Move portions of memory around"
```

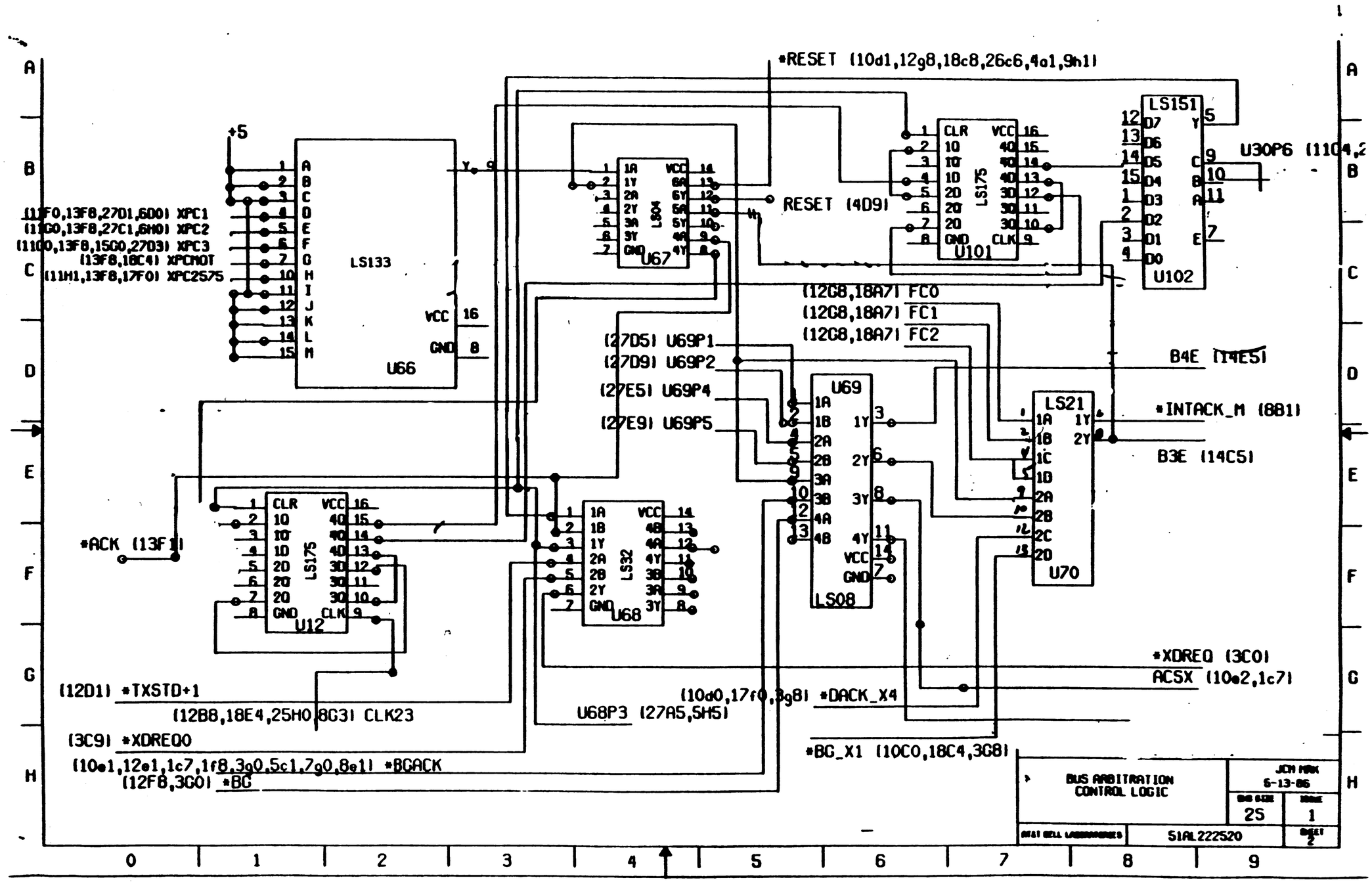
```
.byte 0x0d
.byte 0x0a
.ascii "U     UNIX communication - full-duplex communication with mod
.byte 0x0d
.byte 0x0a
.ascii "W[X] start,end    (hex addresses)"
.byte 0x0d
.byte 0x0a
.ascii "      Write - Upload machine code in Intel hex format to mode
.byte 0x0d
.byte 0x0a
.ascii "                Use the X option to write extended address (32
.byte 0x0d
.byte 0x0a
.byte 0
```

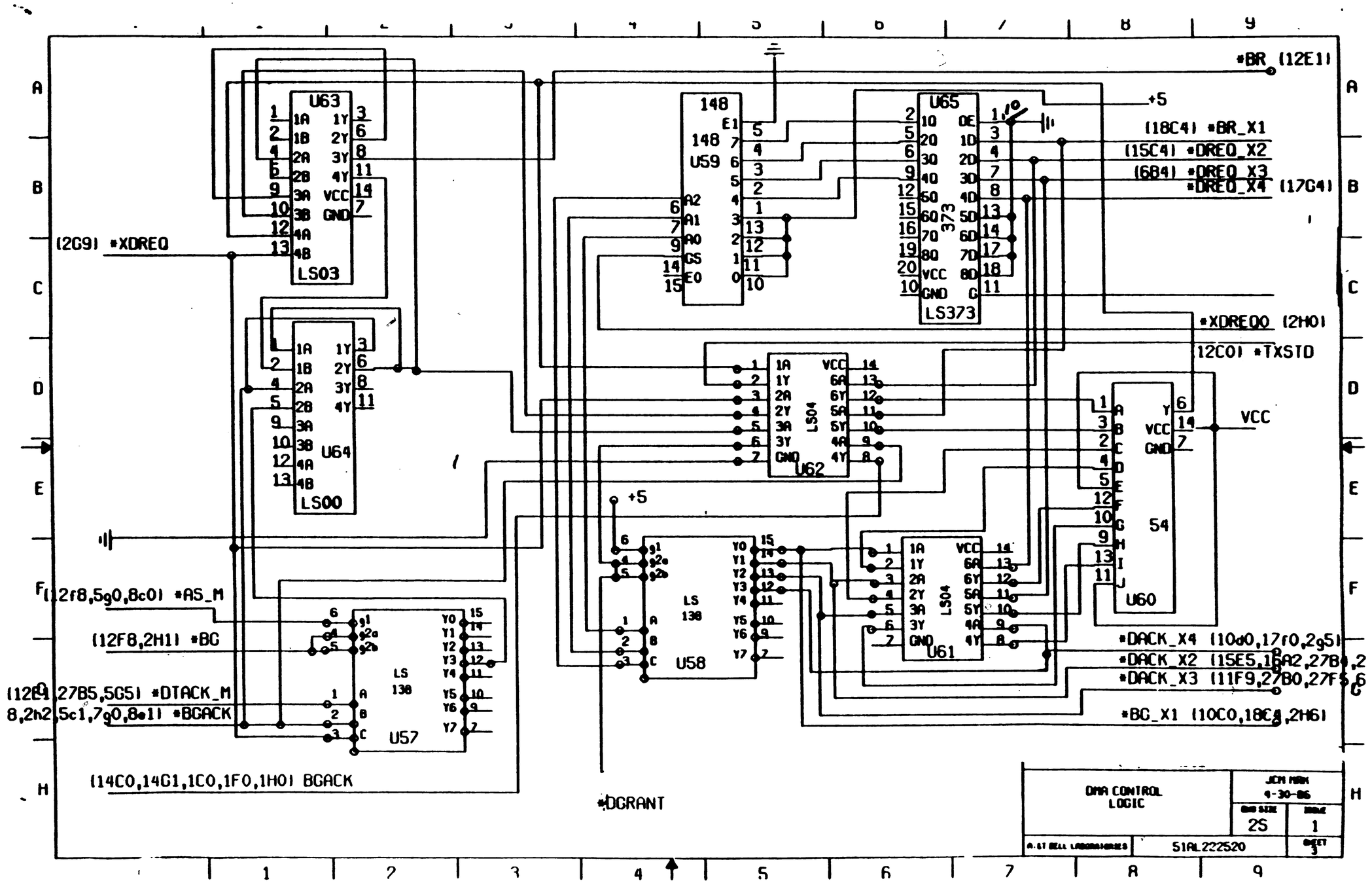
APPENDIX B

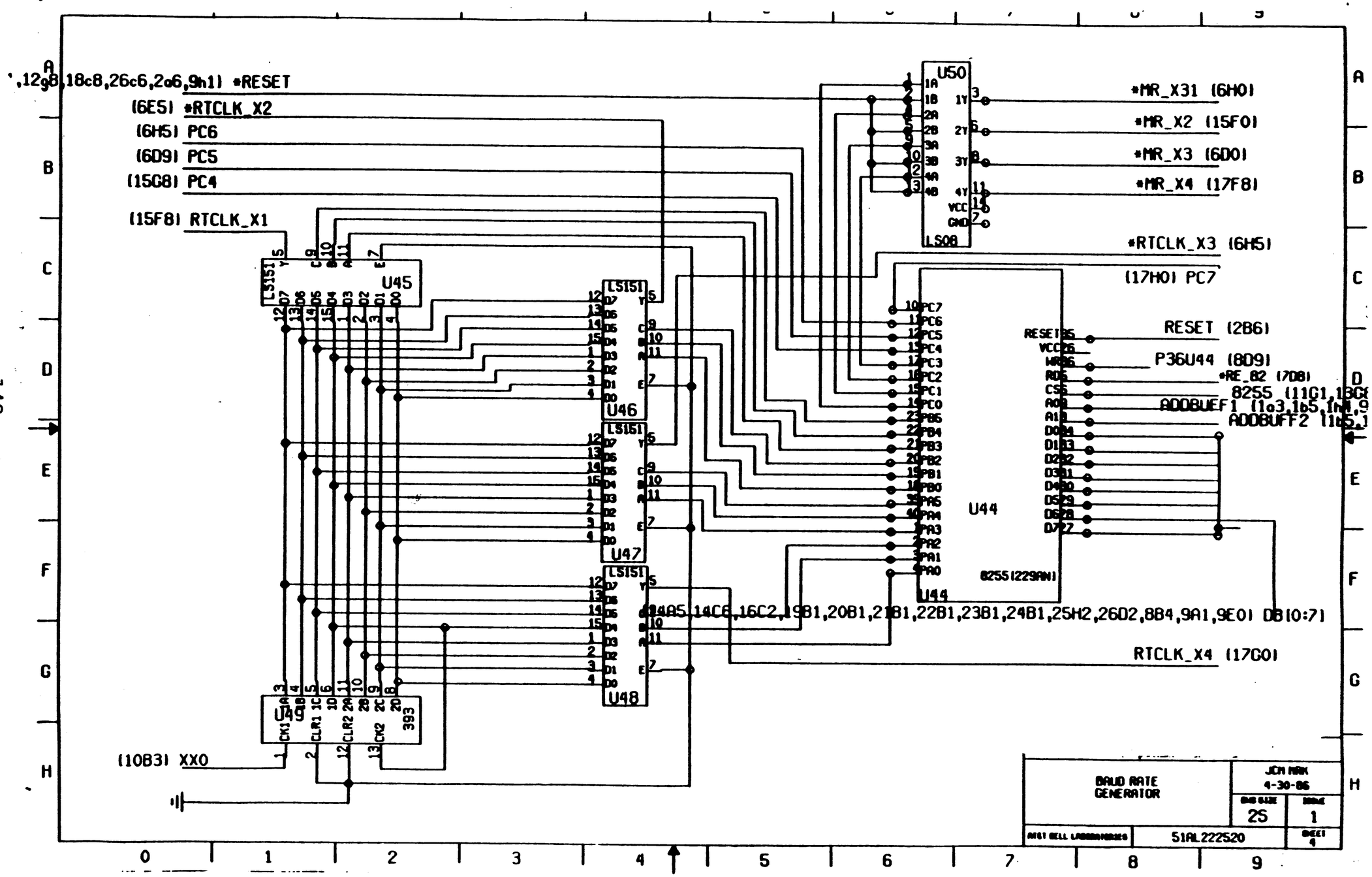
Appendix B contains the schematics for the communication board. These schematics contains all the hardware components in the communication board. The name of the signals are provided as well as off-page references.

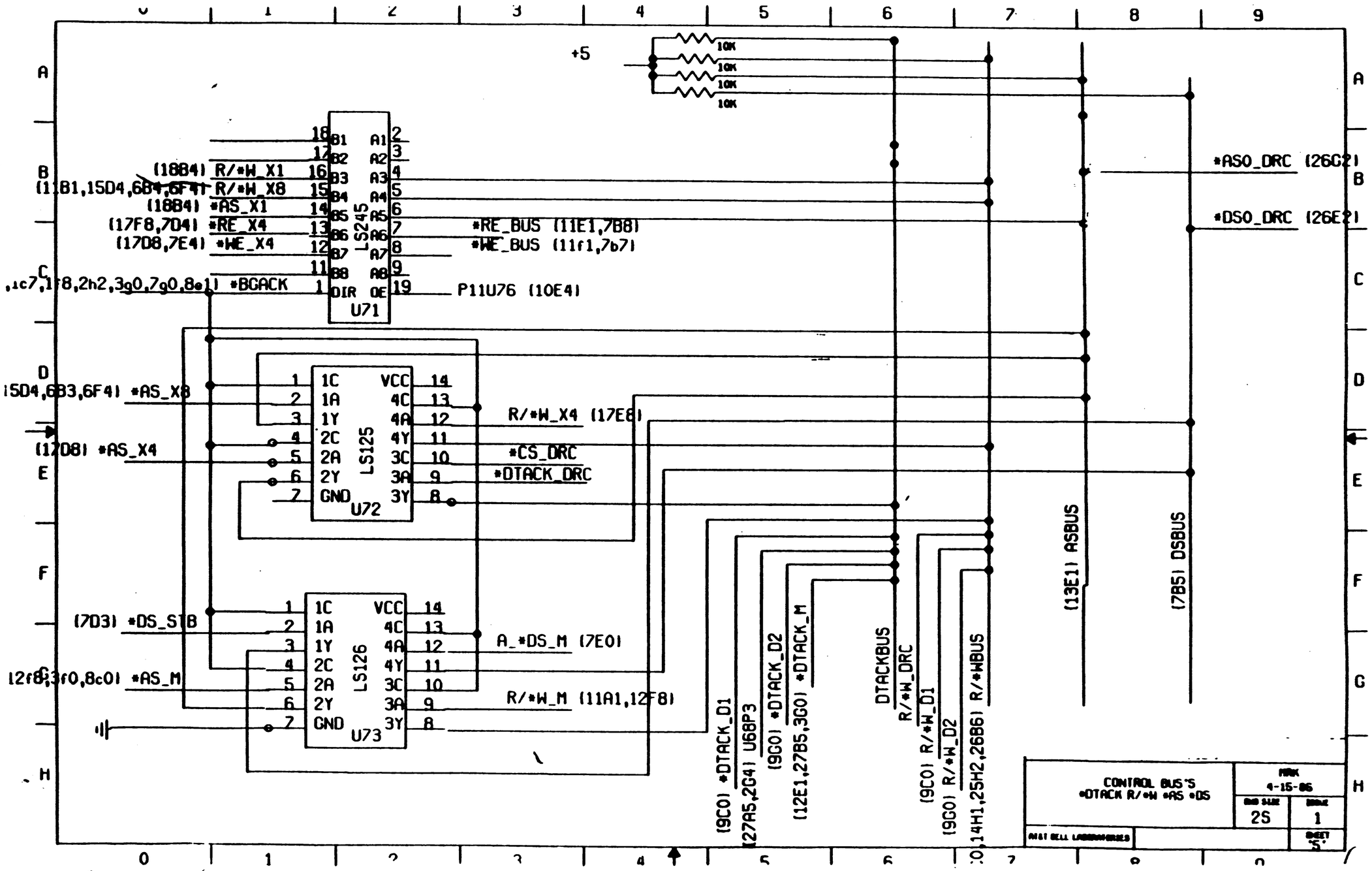

(12C8,8D0) MA1
(12C8,8D0) MA3
(12C8) MA5
(12D8) MA7
(12D8) MA8
(12C8) MA6
(12C8) MA4
(12C8,8D0) MA2
(14C0,14G1,3H1) BGACK
U1
(12D8) MA9
(12D8) MA11
(12D8) MA13
(12E8) MA15
(12E8) MA16
(12D8) MA14
(12D8) MA12
(12D8) MA10
(14C0,14G1,3H1) BGACK
U2
(12E8) MA17
(12E8) MA19
(12E8) MA21
(12F8) MA23
~1,12F8,7d0,8b7) *UDS_M
(12F8) MA22
(12E8) MA20
(12E8) MA18
(14C0,14G1,3H1) BGACK
U3
(4d9,9c0,9g0) ADDBUFF1
(9C0,9G0) ADDBUFF3
ADDBUFF5
ADDBUFF7
ADDBUFF8
ADDBUFF6
ADDBUFF4
(4d9,9c0,9g0) ADDBUFF2
ADDBUFF9
ADDBUFF11
(13G0) ADDBUFF13
(13H1) ADDBUFF15
(13H1,16A7) ADDBUFF16
(13H0) ADDBUFF14
ADDBUFF12
ADDBUFF10
(13D0,16A7) ADDBUFF17
(13D0,16B7) ADDBUFF19
(13c0,16b7) ADDBUFF21
(13A0,16C7) ADDBUFF23
(13C0,16B7) ADDBUFF22
(13C0,16B7) ADDBUFF20
(13D0,16B7) ADDBUFF18
ADDBUFF[0:23]
ADDBUFF0 (9C0,9G0)
ADDBUFF1 (4d9,9c0,9g0)
ADDBUFF2 (4d9,9c0,9g0)
ADDBUFF3 (9C0,9G0)
ADDBUFF4
ADDBUFF5
ADDBUFF6
ADDBUFF7
LS245
U31
XA0 (11B1,11C1,15B0,17A3,6A0)
XA1 (15B0,17A3,18C1,6A0)
XA2 (15B0,17A3,18D1,6A0)
XA3 (15B0,17B3,18D1,6A0)
XA4 (15C0,17B3,18D1,6A0)
XA5 (15C0,17B3,18D1,6B0)
XA6 (15C0,17B3,18D1,6B0)
XA7 (15C0,17B3,18D1,6B0)
*BGACK (10e1,12e1,2h2,3g0,5c1,7g0,8e1)
ACSX (10e2,2g8)
ADDBUFF8
ADDBUFF10
ADDBUFF12
ADDBUFF14 (13H0)
ADDBUFF15 (13H1)
ADDBUFF13 (13C0)
ADDBUFF11
ADDBUFF9
U32
XA8 (15C0,17C3,18D1,6B0)
XA10 (15D0,17C3,18E1,6B0)
XA12 (15D0,17C3,18E1,6B0)
XA14 (15D0,17D3,18E1,6C0)
XA15 (15D0,17D3,18F1,6C0)
XA13 (15D0,17D3,18E1,6C0)
XA11 (15D0,17C3,18E1,6B0)
XA9 (15C0,17C3,18E1,6B0)
*BGACK (10e1,12e1,2h2,3g0,5c1,7g0,8e1)
ADDBUFF16 (13H1,16A7)
ADDBUFF17 (13D0,16A7)
ADDBUFF18 (13D0,16B7)
ADDBUFF19 (13D0,16B7)
ADDBUFF20 (13C0,16B7)
ADDBUFF21 (13c0,16b7)
ADDBUFF22 (13C0,16B7)
ADDBUFF23 (13A0,16C7)
(10D5,14C5) B36E
U36
XA16 (17D3,18F1)
XA17 (17D3,18F1)
XA18 (17E3,18F1)
XA19 (17E3,18F1)
XA20 (17E3,18F1)
XA21 (17E3,18F1)
XA22 (17E3,18G1)
XA23 (17F3,18G1)
ADDRESS BUFFERS
MRK JCN 4-30-85
SIZE 2S
DWG NO 1
51AL222520
SHEET 1
0 1 2 3 4 5 6 7 8 9
A B C D E F G H

145.

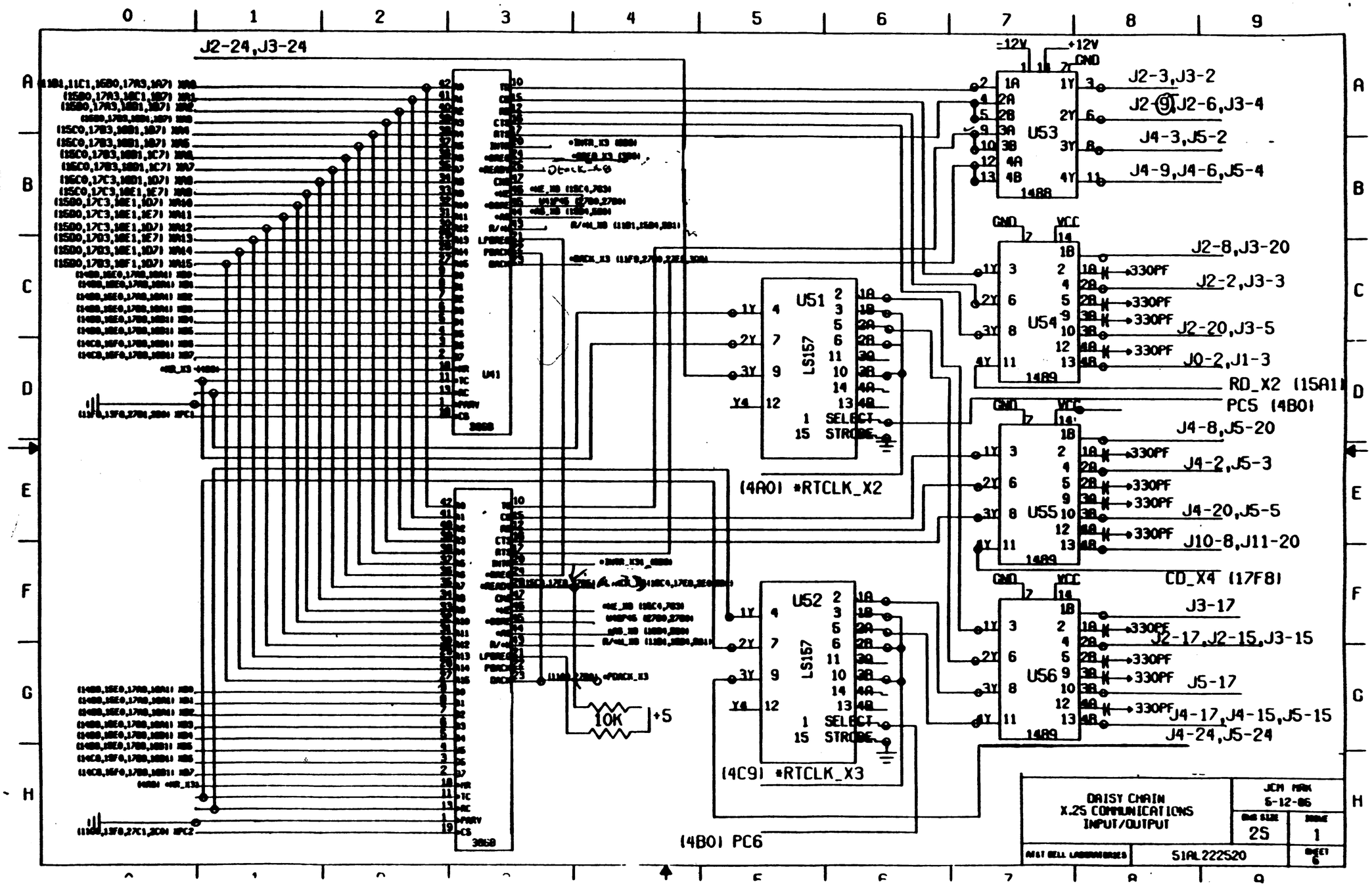

*RESET (10d1,12g8,18c8,26c6,4a1,9h1)
+5
(11F0,13F8,27D1,6D0) XPC1
(11C0,13F8,27C1,6H0) XPC2
(11D0,13F8,15C0,27D3) XPC3
(13F8,18C4) XPCMOT
(11H1,13F8,17F0) XPC2575
LS133
U66
VCC 16
GND 8
U67
LS04
RESET (4D9)
U101
LS175
CLR
GND
CLK
LS151
U102
U30P6 (11C4,2
(12C8,18A7) FC0
(12C8,18A7) FC1
(12C8,18A7) FC2
B4E (14E5)
(27D5) U69P1
(27D9) U69P2
(27E5) U69P4
(27E9) U69P5
U69
LS08
LS21
U70
*INTACK_M (8B1)
B3E (14C5)
*ACK (13F1)
U12
U68
LS32
U68P3 (27A5,5H5)
*XDREQ (3C0)
ACSX (10e2,1c7)
(10d0,17f0,3g8) *DACK_X4
(12D1) *TXSTD+1
(12B8,18E4,25H0,8G3) CLK23
(3C9) *XDREQ0
(10e1,12e1,1c7,1f8,3g0,5c1,7g0,8e1) *BGACK
(12F8,3G0) *BG
*BG_X1 (10C0,18C4,3G8)
BUS ARBITRATION CONTROL LOGIC
JCH MRK 6-13-86
BUS SIZE 2S
SIZE 1
SIAL222520
SHEET 2


*BR (12E1)
U63
LS03
(2G9) *XDREQ
U64
LS00
148
U59
148
U65
LS373
+5
(18C4) *BR_X1
(15C4) *DREQ_X2
(6B4) *DREQ_X3
*DREQ_X4 (17G4)
*XDREQ0 (2H0)
12C0) *TXSTD
U62
LS04
VCC
U61
LS04
54
U60
+5
LS 138
U58
LS 138
U57
(12f8,5g0,8c0) *AS_M
(12F8,2H1) *BC
(12E1,27B5,5G5) *DTACK_M
8,2h2,5c1,7g0,8e1) *BGACK
(14C0,14G1,1C0,1F0,1H0) BGACK
*DGRANT
*DACK_X4 (10d0,17f0,2g5)
*DACK_X2 (15E5,16A2,27B0,2
*DACK_X3 (11F9,27B0,27F5,6
*BG_X1 (10C0,18C8,2H6)
DMA CONTROL LOGIC
JCM MBM 9-30-86
SIZE 2S
DWG 1
SIAL222520
SHEET 3

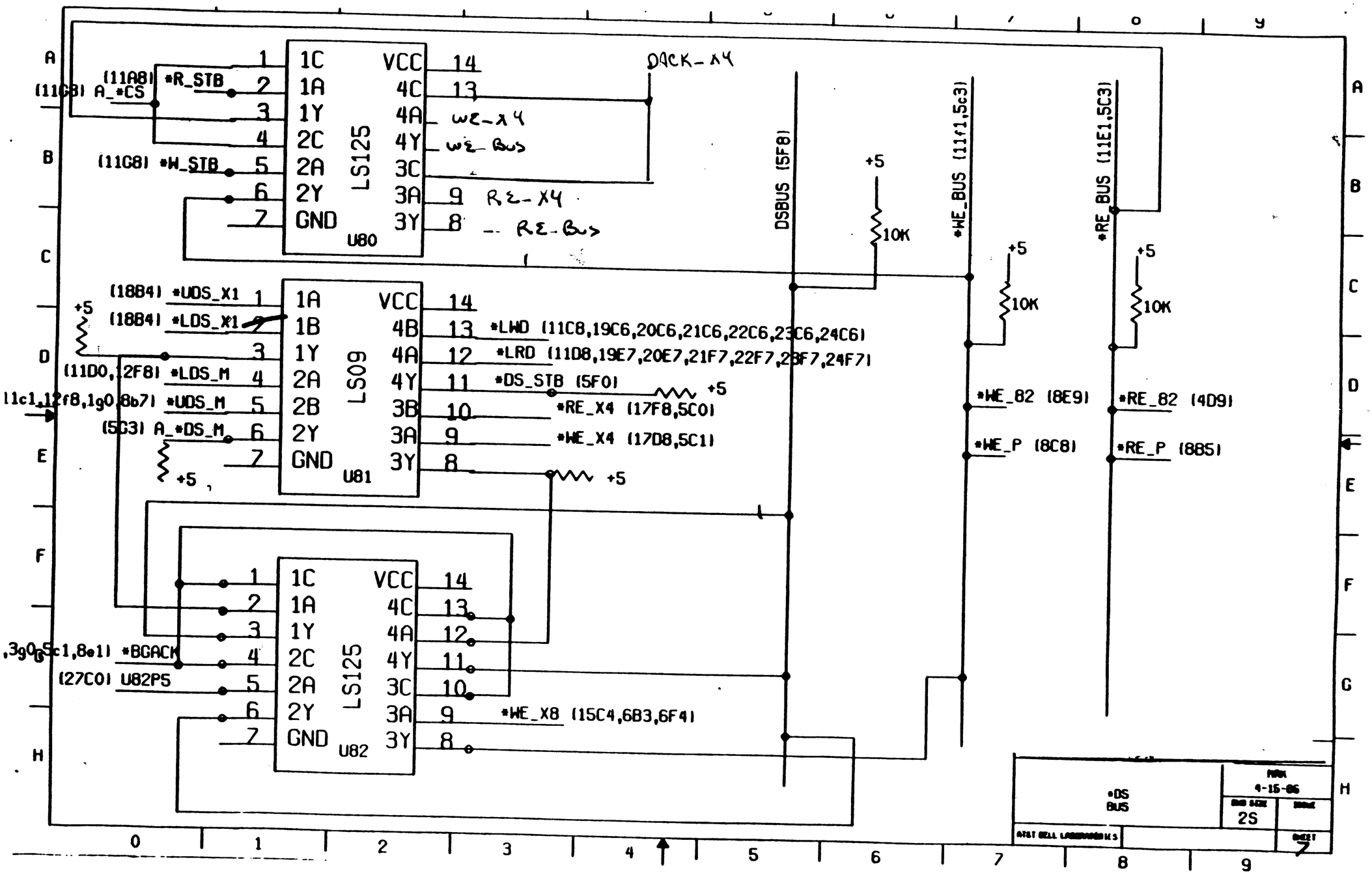

*,12g8,18c8,26c6,2o6,9h1) *RESET
(6E5) *RTCLK_X2
(6H5) PC6
(6D9) PC5
(15G8) PC4
(15F8) RTCLK_X1
U45
U46
U47
U48
U49
U50
U44
LS08
LS151
74393
*MR_X31 (6H0)
*MR_X2 (15F0)
*MR_X3 (6D0)
*MR_X4 (17F8)
*RTCLK_X3 (6H5)
(17H0) PC7
RESET (2B6)
P36U44 (8D9)
*RE_82 (7D8)
8255 (11C1,13C
ADDBUEF1 (1o3,1b5,1n1,9
ADDBUFF2 (1b5,1
8255(229AN)
4A5,14C6,16C2,19B1,20B1,21B1,22B1,23B1,24B1,25H2,26D2,8B4,9A1,9E0) DB(0:7)
RTCLK_X4 (17G0)
(10B3) XX0
BAUD RATE GENERATOR
JCM MRK 4-30-86
25
1
51AL222520

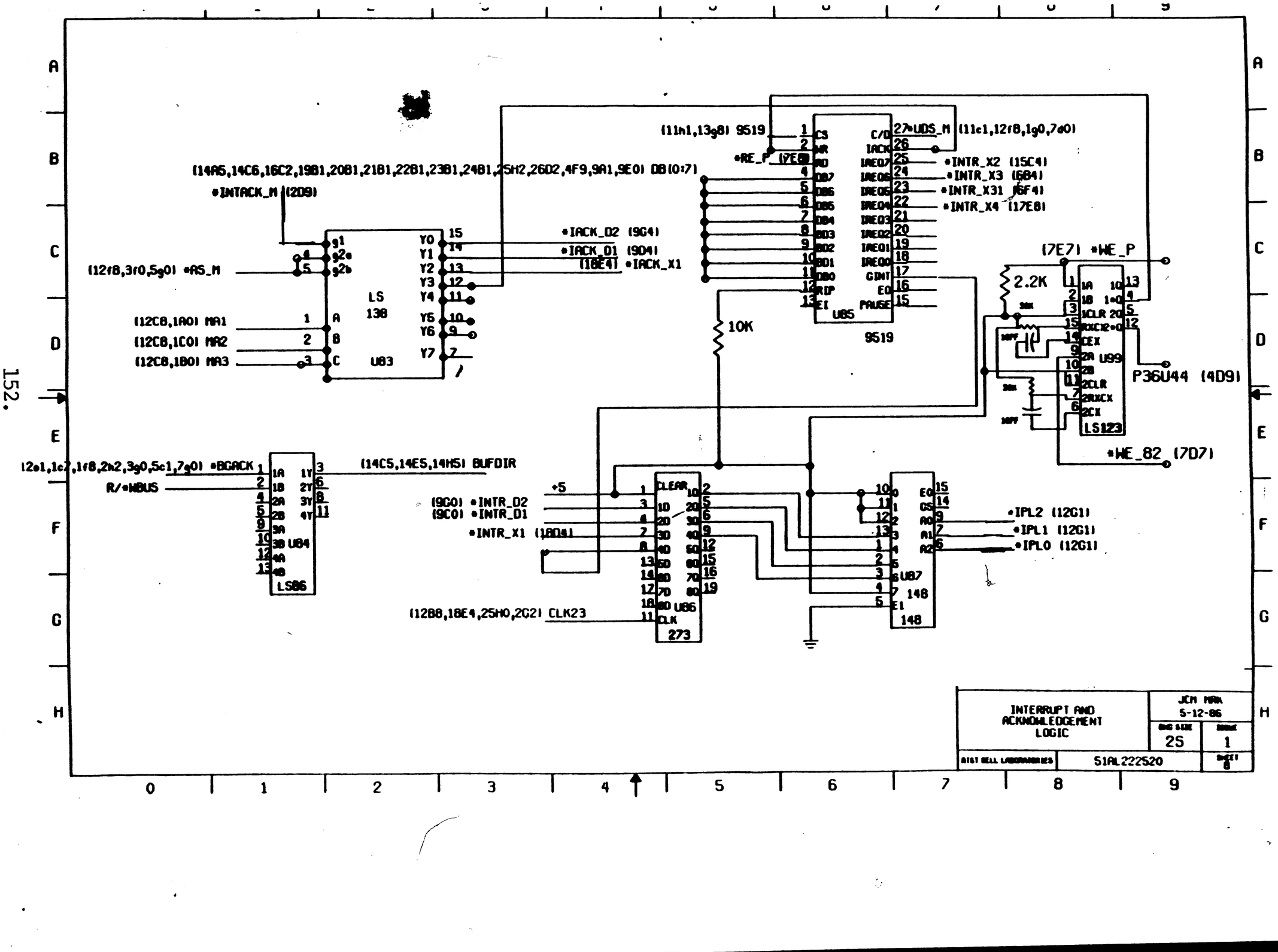

+5
10K
10K
10K
10K
U71
LS245
18 B1
17 B2
(18B4) R/*W_X1 16 B3
(11B1,15D4,6B4,6F4) R/*W_X8 15 B4
(18B4) *AS_X1 14 B5
(17F8,7D4) *RE_X4 13 B6
(17D8,7E4) *WE_X4 12 B7
11 B8
(1c7,1f8,2h2,3g0,7g0,8e1) *BGACK 1 DIR
A1 2
A2 3
A3 4
A4 5
A5 6
A6 7
A7 8
A8 9
OE 19
*RE_BUS (11E1,7B8)
*WE_BUS (11f1,7b7)
P11U76 (10E4)
*ASO_DRC (26C2)
*DSO_DRC (26E2)
U72
LS125
(15D4,6B3,6F4) *AS_X8
(17D8) *AS_X4
1C
1A
1Y
2C
2A
2Y
GND
VCC
4C
4A
4Y
3C
3A
3Y
14
13
12
11
10
9
8
R/*W_X4 (17E8)
*CS_DRC
*DTACK_DRC
U73
LS126
(7D3) *DS_STB
(12f8,13f0,8c0) *AS_M
A_*DS_M (7E0)
R/*W_M (11A1,12F8)
(9C0) *DTACK_D1
(27A5,2G4) U68P3
(9G0) *DTACK_D2
(12E1,27B5,3G0) *DTACK_M
DTACKBUS
R/*W_DRC
(9C0) R/*W_D1
(9G0) R/*W_D2
(10,14H1,25H2,26B6) R/*WBUS
(13E1) ASBUS
(7B5) DSBUS
CONTROL BUS'S
*DTACK R/*W *AS *DS
MRK
4-15-86
SIZE
2S
DRAWING
1
SHEET
5
0 1 2 3 4 5 6 7 8 9
A B C D E F G H

149


J2-24,J3-24
-12V
+12V
GND
J2-3,J3-2
J2-9,J2-6,J3-4
J4-3,J5-2
J4-9,J4-6,J5-4
U53
1488
GND
VCC
J2-8,J3-20
J2-2,J3-3
J2-20,J3-5
J0-2,J1-3
330PF
U54
1489
RD_X2 (15A1)
PCS (4B01
J4-8,J5-20
J4-2,J5-3
J4-20,J5-5
J10-8,J11-20
U55
CD_X4 (17F8)
J3-17
J2-17,J2-15,J3-15
J5-17
J4-17,J4-15,J5-15
J4-24,J5-24
U56
U51
U52
LS157
SELECT
STROBE
(4A01 *RTCLK_X2
(4C9) *RTCLK_X3
(4B01 PC6
10K
+5
DAISY CHAIN
X.25 COMMUNICATIONS
INPUT/OUTPUT
JCM MRK
6-12-86
SIZE
2S
SHEET
1
SIAL222520
SHEET
6

150.


U80
LS125
1C
1A
1Y
2C
2A
2Y
GND
VCC
4C
4A
4Y
3C
3A
3Y
(11A8) *R_STB
(11G8) A_*CS
(11G8) *W_STB
DACK-X4
WE-X4
WE_Bus
RE-X4
RE-Bus
U81
LS09
1A
1B
1Y
2A
2B
2Y
GND
VCC
4B
4A
4Y
3B
3A
3Y
(18B4) *UDS_X1
(18B4) *LDS_X1
(11D0,12F8) *LDS_M
(11c1,12f8,1g0,8b7) *UDS_M
(5C3) A_*DS_M
+5
*LWD (11C8,19C6,20C6,21C6,22C6,23C6,24C6)
*LRD (11D8,19E7,20E7,21F7,22F7,23F7,24F7)
*DS_STB (5F0)
*RE_X4 (17F8,5C0)
*WE_X4 (17D8,5C1)
U82
LS125
(3g0,5c1,8e1) *BGACK
(27C0) U82P5
*WE_X8 (15C4,6B3,6F4)
DSBUS (5F8)
10K
*WE_BUS (11f1,5c3)
*RE_BUS (11E1,5C3)
*WE_82 (8E9)
*RE_82 (4D9)
*WE_P (8C8)
*RE_P (8B5)
*DS BUS
4-15-86
2S


(14A5,14C6,16C2,19B1,20B1,21B1,22B1,23B1,24B1,25H2,26D2,4F9,9A1,9E0) DB(0:7)
*INTACK_M (2D9)
(12f8,3f0,5g0) *AS_M
(12C8,1A0) MA1
(12C8,1C0) MA2
(12C8,1B0) MA3
LS 138
U83
*IACK_D2 (9G4)
*IACK_D1 (9D4)
(16E4) *IACK_X1
(11h1,13g8) 9519
*RE_P (7E8)
U85
9519
*UDS_M (11c1,12f8,1g0,7d0)
*INTR_X2 (15C4)
*INTR_X3 (6B4)
*INTR_X31 (6F4)
*INTR_X4 (17E8)
10K
(7E7) *WE_P
2.2K
U99
LS123
P36U44 (4D9)
*WE_82 (7D7)
(12e1,1c7,1f8,2h2,3g0,5c1,7g0) *BGACK
R/*WBUS
U84
LS86
(14C5,14E5,14H5) BUFDIR
+5
(9C0) *INTR_D2
(9C0) *INTR_D1
*INTR_X1 (18D4)
(12B8,18E4,25H0,2G2) CLK23
U86
273
U87
148
*IPL2 (12G1)
*IPL1 (12G1)
*IPL0 (12G1)
INTERRUPT AND ACKNOWLEDGEMENT LOGIC
JCM MRA
5-12-86
25
1
BELL LABORATORIES
51AL222520
SHEET 8

DB[0:7] (1B1,2SH2,2602,4F9,8B4)

DB[0:7] (14A5,14C6,16C2,19B1,20B1,21B1,22B1,23B1,24B1,25H2,2602,4F9,8B4)

(1B3,1B5,1H4) BUFF[0:3]

U19 68681

U20 68681

*IACK_D1 (8C4)

*IACK_D2 (8C4)

COMCLK (10A1)

*RESET (10d1,12g8,18 8,26c6,2a6,4a1)

U14 1488 U15 1489 U16 1488 U17 1489 U18 1489

J8-6 J9-2 J9-4 J8-3 J8-20 J7-4 J8-2 J8-4 J9-8 J9-3 J9-5 J7-20 J7-3 J7-6 J6-3 J6-6 J7-2 J6-2 J6-20 J6-4

330PF

DATA COMMUNICATIONS TO THE TUTOR BOARD

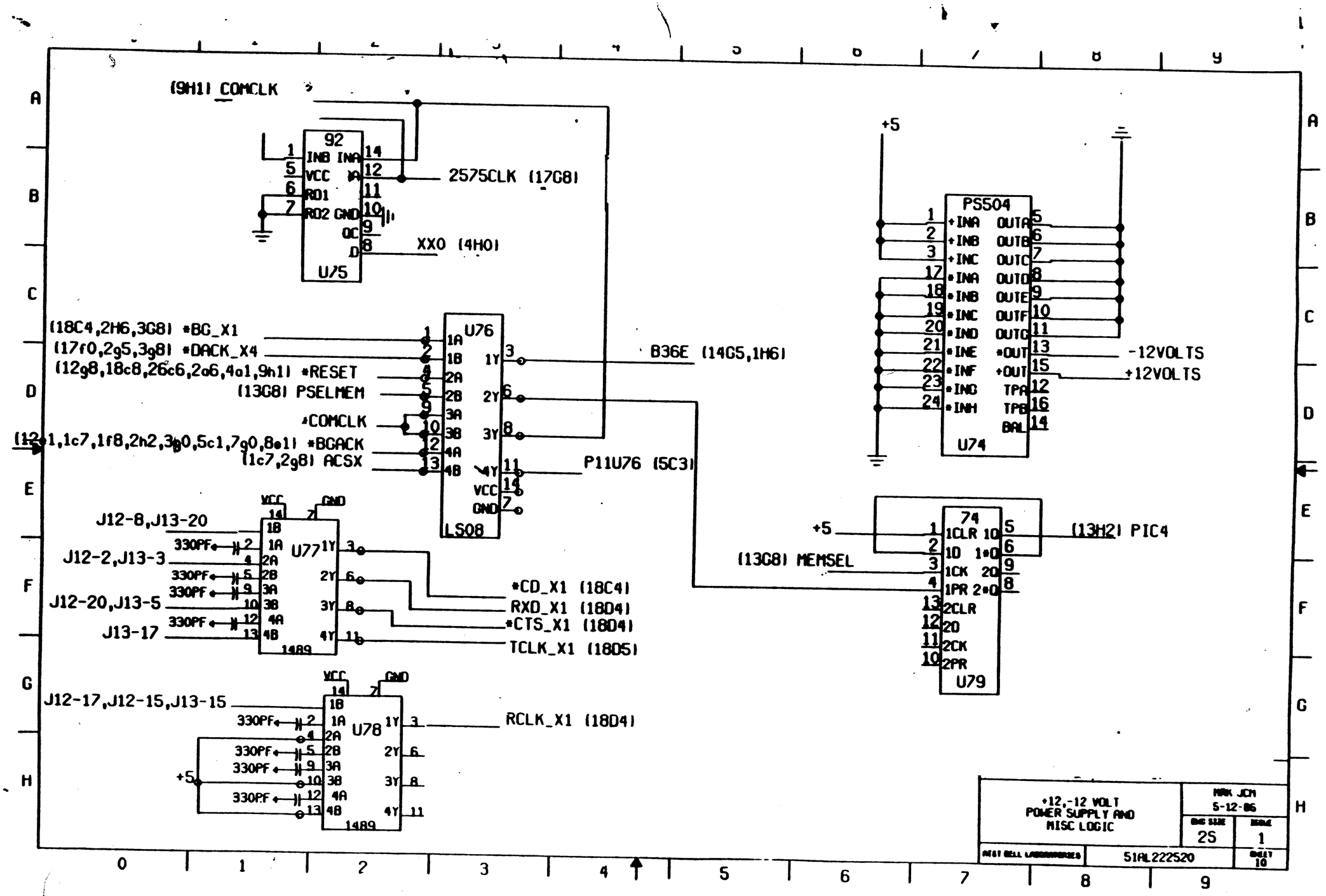
JCM MRK 4-30-86

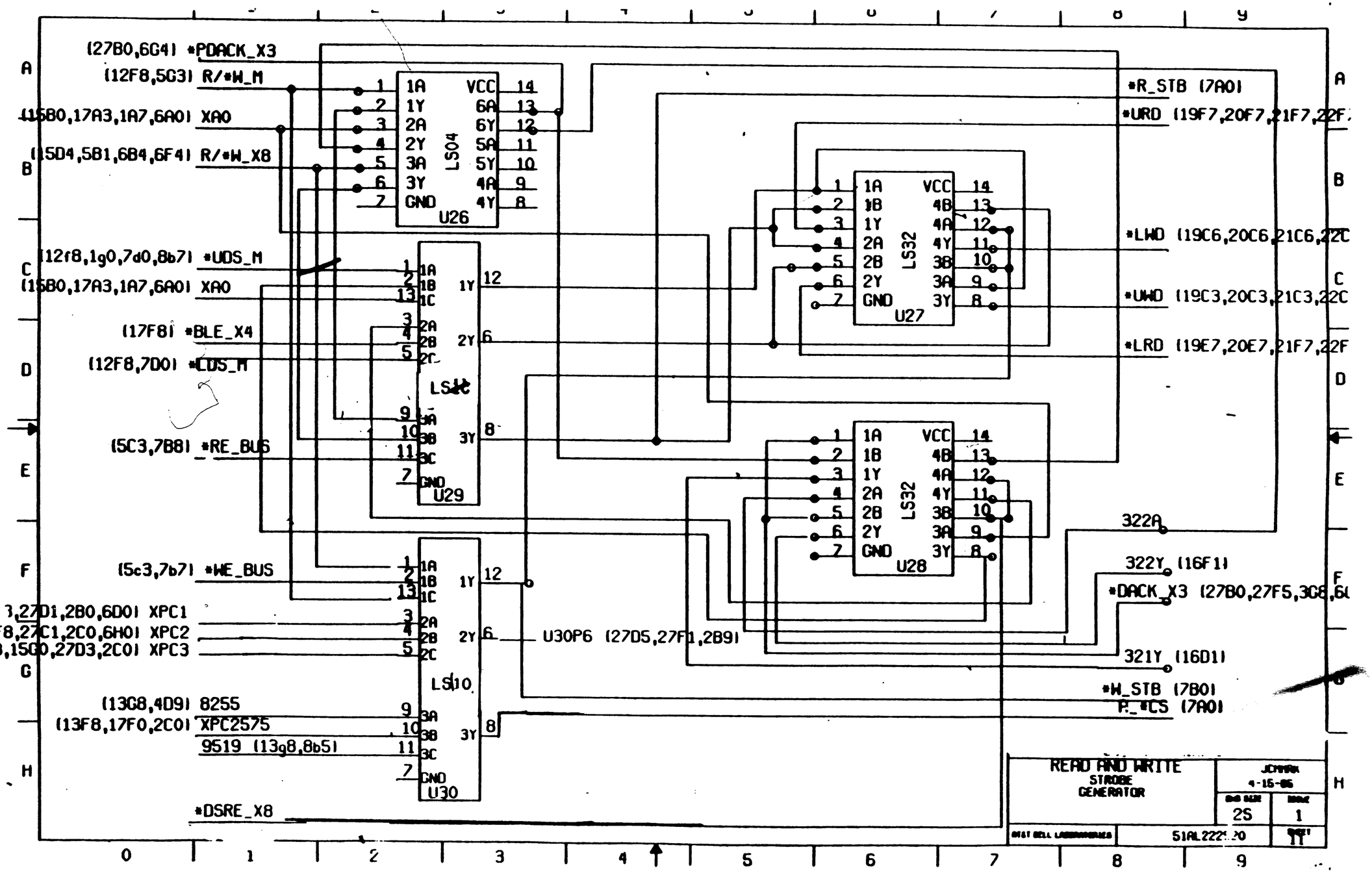
SIZE 2S

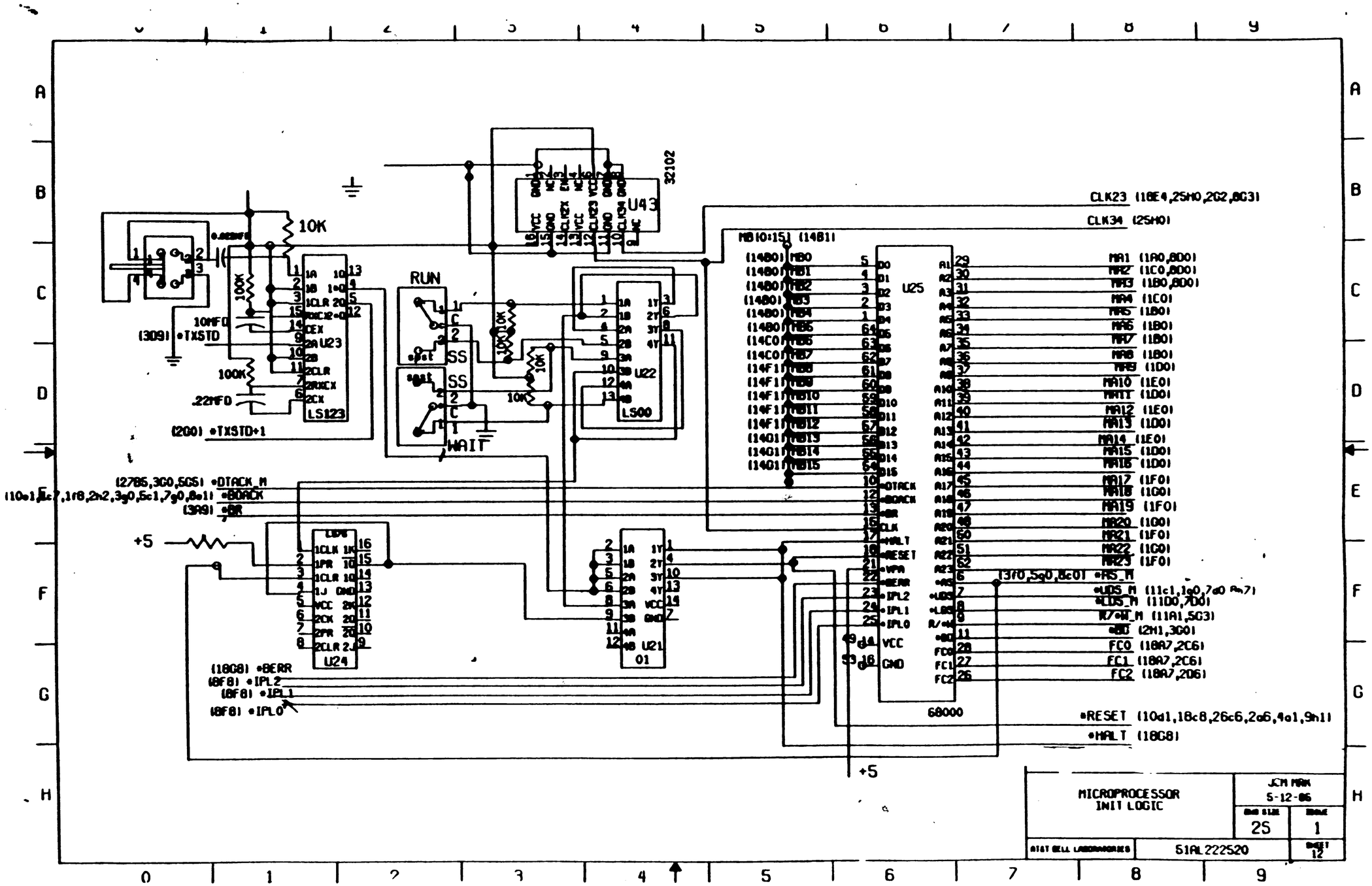
DWG 1

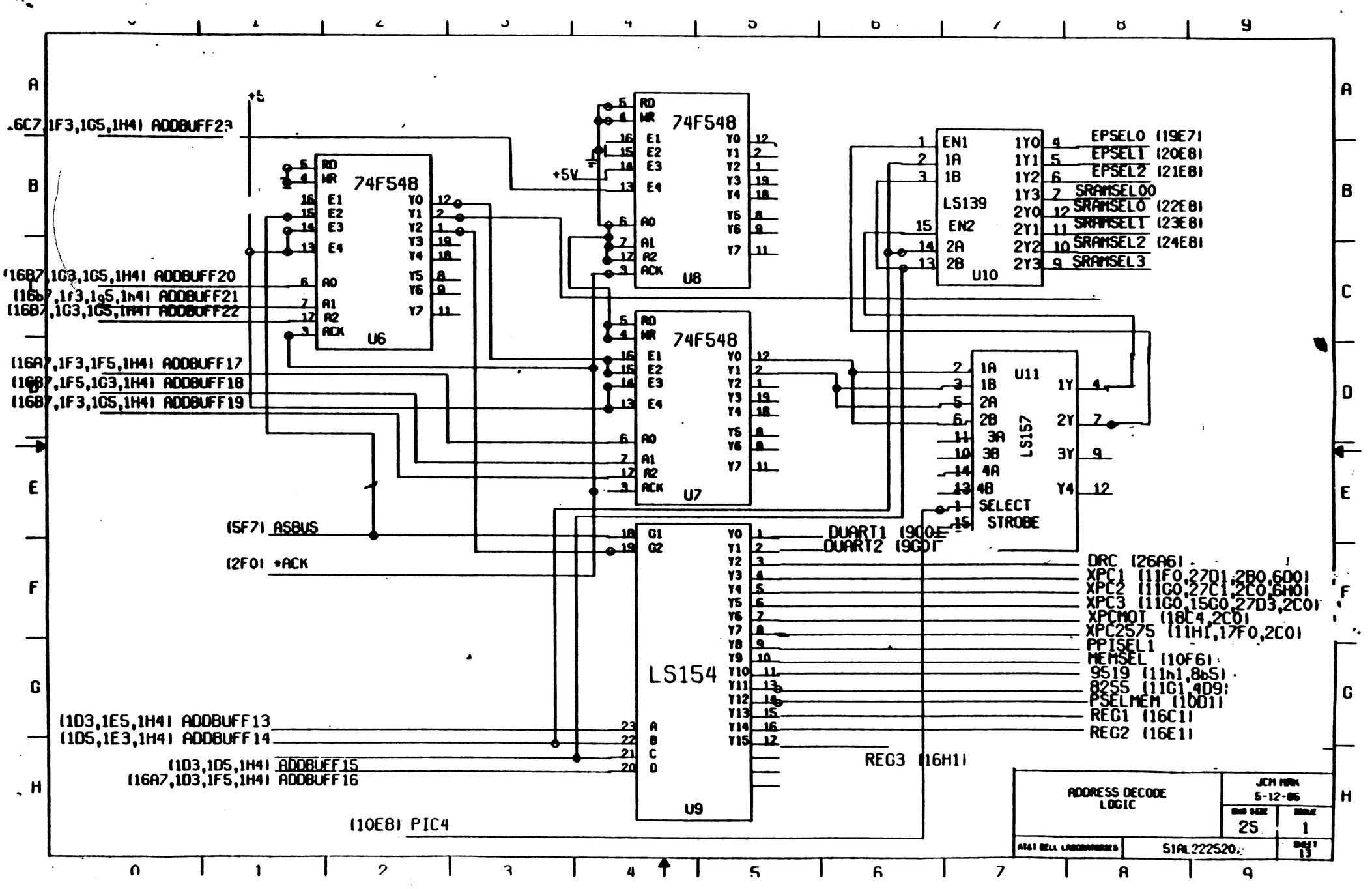
SIAL222520

SHEET 9


(9H1) COMCLK
92
INB INA
VCC
RD1
RD2 GND
QC
D
U75
2575CLK (17G8)
XXO (4HO)
(18C4,2H6,3G8) *BG_X1
(17f0,2g5,3g8) *DACK_X4
(12g8,18c8,26c6,2o6,4o1,9h1) *RESET
(13G8) PSELMEM
*COMCLK
(12g1,1c7,1f8,2h2,3g0,5c1,7g0,8e1) *BGACK
(1c7,2g8) ACSX
U76
1A 1B 1Y 2A 2B 2Y 3A 3B 3Y 4A 4B 4Y
VCC GND
LS08
B36E (14G5,1H6)
P11U76 (5C3)
J12-8,J13-20
J12-2,J13-3
J12-20,J13-5
J13-17
330PF
U77
1489
*CD_X1 (18C4)
RXD_X1 (18D4)
*CTS_X1 (18D4)
TCLK_X1 (18D5)
J12-17,J12-15,J13-15
U78
RCLK_X1 (18D4)
+5
PS504
+INA +INB +INC +IND +INE +INF +ING +INH
OUTA OUTB OUTC OUTD OUTE OUTF OUTG
*OUT +OUT TPA TPB BAL
U74
-12VOLTS
+12VOLTS
74
1CLR 1D 1CK 1PR 2CLR 2D 2CK 2PR
1*Q 2*Q
U79
(13H2) PIC4
(13G8) MEMSEL
+12,-12 VOLT
POWER SUPPLY AND
MISC LOGIC
MARK JCH
5-12-86
SIZE 2S
DWG 1
SHEET 10
51AL222520


(27B0,6G4) *PDACK_X3
(12F8,5G3) R/*W_M
(15B0,17A3,1A7,6A0) XA0
(15D4,5B1,6B4,6F4) R/*W_X8
(12f8,1g0,7d0,8b7) *UDS_M
(15B0,17A3,1A7,6A0) XA0
(17F8) *BLE_X4
(12F8,7D0) *LDS_M
(5C3,7B8) *RE_BUS
(5c3,7b7) *WE_BUS
3,27D1,2B0,6D0) XPC1
F8,27C1,2C0,6H0) XPC2
8,15G0,27D3,2C0) XPC3
(13G8,4D9) 8255
(13F8,17F0,2C0) XPC2575
9519 (13g8,8b5)
*DSRE_X8
U26
U27
U28
U29
U30
LS04
LS32
LS10
U30P6 (27D5,27F1,2B9)
*R_STB (7A0)
*URD (19F7,20F7,21F7,22F
*LWD (19C6,20C6,21C6,22C
*UWD (19C3,20C3,21C3,22C
*LRD (19E7,20E7,21F7,22F
322A
322Y (16F1)
*DACK_X3 (27B0,27F5,3C8,6L
321Y (16D1)
*W_STB (7B0)
P_*CS (7A0)
READ AND WRITE
STROBE
GENERATOR
JCHAN
4-15-85
2S
1
51AL222


32102
U43
10K
RUN
WAIT
SS
U23
LS123
U22
LS00
U24
LS96
U21
01
68000
U25
+5
CLK23 (18E4,25H0,2G2,8G3)
CLK34 (25H0)
MB(0:15) (14B1)
MA1 (1A0,8D0)
MA2 (1C0,8D0)
MA3 (1B0,8D0)
MA4 (1C0)
MA5 (1B0)
MA6 (1B0)
MA7 (1B0)
MA8 (1B0)
MA9 (1D0)
MA10 (1E0)
MA11 (1D0)
MA12 (1E0)
MA13 (1D0)
MA14 (1E0)
MA15 (1D0)
MA16 (1D0)
MA17 (1F0)
MA18 (1G0)
MA19 (1F0)
MA20 (1G0)
MA21 (1F0)
MA22 (1G0)
MA23 (1F0)
(309) *TXSTD
(200) *TXSTD+1
(2785,3G0,5G5) *DTACK_M
(3991) *BR
(18C8) *BERR
(8F8) *IPL2
(8F8) *IPL1
(8F8) *IPL0
*RESET (10d1,18c8,26c6,2a6,4a1,9h1)
*HALT (18C8)
MICROPROCESSOR
INIT LOGIC
JCM MAR
5-12-86
SIZE 2S
DRAWING 1
SHEET 12
AT&T BELL LABORATORIES
51AL222520


74F548
U6
U7
U8
74F548
LS154
U9
LS139
U10
LS157
U11
+5
+5V
(16B7,1G3,1G5,1H4) ADDBUFF20
(16B7,1F3,1G5,1H4) ADDBUFF21
(16B7,1G3,1G5,1H4) ADDBUFF22
(16A7,1F3,1F5,1H4) ADDBUFF17
(16B7,1F5,1G3,1H4) ADDBUFF18
(16B7,1F3,1G5,1H4) ADDBUFF19
(5F7) ASBUS
(2F0) *ACK
(1D3,1E5,1H4) ADDBUFF13
(1D5,1E3,1H4) ADDBUFF14
(1D3,1D5,1H4) ADDBUFF15
(16A7,1D3,1F5,1H4) ADDBUFF16
(10E8) PIC4
EPSEL0 (19E7)
EPSEL1 (20E8)
EPSEL2 (21E8)
SRAMSEL00
SRAMSEL0 (22E8)
SRAMSEL1 (23E8)
SRAMSEL2 (24E8)
SRAMSEL3
DUART1 (9C0E)
DUART2 (9C0F)
DRC (26A6)
XPC1 (11F0,27D1,2B0,6D0)
XPC2 (11G0,27C1,2C0,6H0)
XPC3 (11G0,15G0,27D3,2C0)
XPCMOT (18C4,2C0)
XPC2575 (11H1,17F0,2C0)
PPISEL1
MEMSEL (10F6)
9519 (11H1,8B5)
8255 (11G1,4D9)
PSELMEM (10D1)
REG1 (16C1)
REG2 (16E1)
REG3 (16H1)
ADDRESS DECODE LOGIC
JCM MRK 5-12-86
2S
1
51AL222520.
13

157.

DB[0:15] (19C1,20C1,21C1,22C1,23C1,24C1,25H2)

U4 LS245
MB0 (12B5,12C5) 2
MB1 (12B5,12C5) 3
MB2 (12B5,12C5) 4
MB3 (12B5,12C5) 5
MB4 (12B5,12C5) 6
MB5 (12B5,12C5) 7
MB6 (12B5,12D5) 8
MB7 (12B5,12D5) 9
(1H0,3H1) BGACK 19
R/*WBUS (25H2,26B6,5G7)
A1 A2 A3 A4 A5 A6 A7 A8 OE DIR
B1 B2 B3 B4 B5 B6 B7 B8 1
18 17 16 15 14 13 12 11
(16A2,19A1,20A1,21A1,22A1,23A1,24A1,25H2,26C2,4F9,8B4,9A1,9E0) DB0
(16B2,19A1,20A1,21A1,22A1,23A1,24A1,25H2,26C2,4F9,8B4,9A1,9E0) DB1
(16B2,19B1,20B1,21B1,22B1,23B1,24B1,25H2,26C2,4F9,8B4,9A1,9E0) DB2
(16B2,19B1,20B1,21B1,22B1,23B1,24B1,25H2,26D2,4F9,8B4,9A1,9E0) DB3
(16B2,19B1,20B1,21B1,22B1,23B1,24B1,25H2,26D2,4F9,8B4,9A1,9E0) DB4
(16B2,19B1,20B1,21B1,22B1,23B1,24B1,25H2,26D2,4F9,8B4,9A1,9E0) DB5
(16C2,19B1,20B1,21B1,22B1,23B1,24B1,25H2,26D2,4F9,8B4,9A1,9E0) DB6
(16C2,19B1,20B1,21B1,22B1,23B1,24B1,25H2,26D2,4F9,8B4,9A1,9E0) DB7

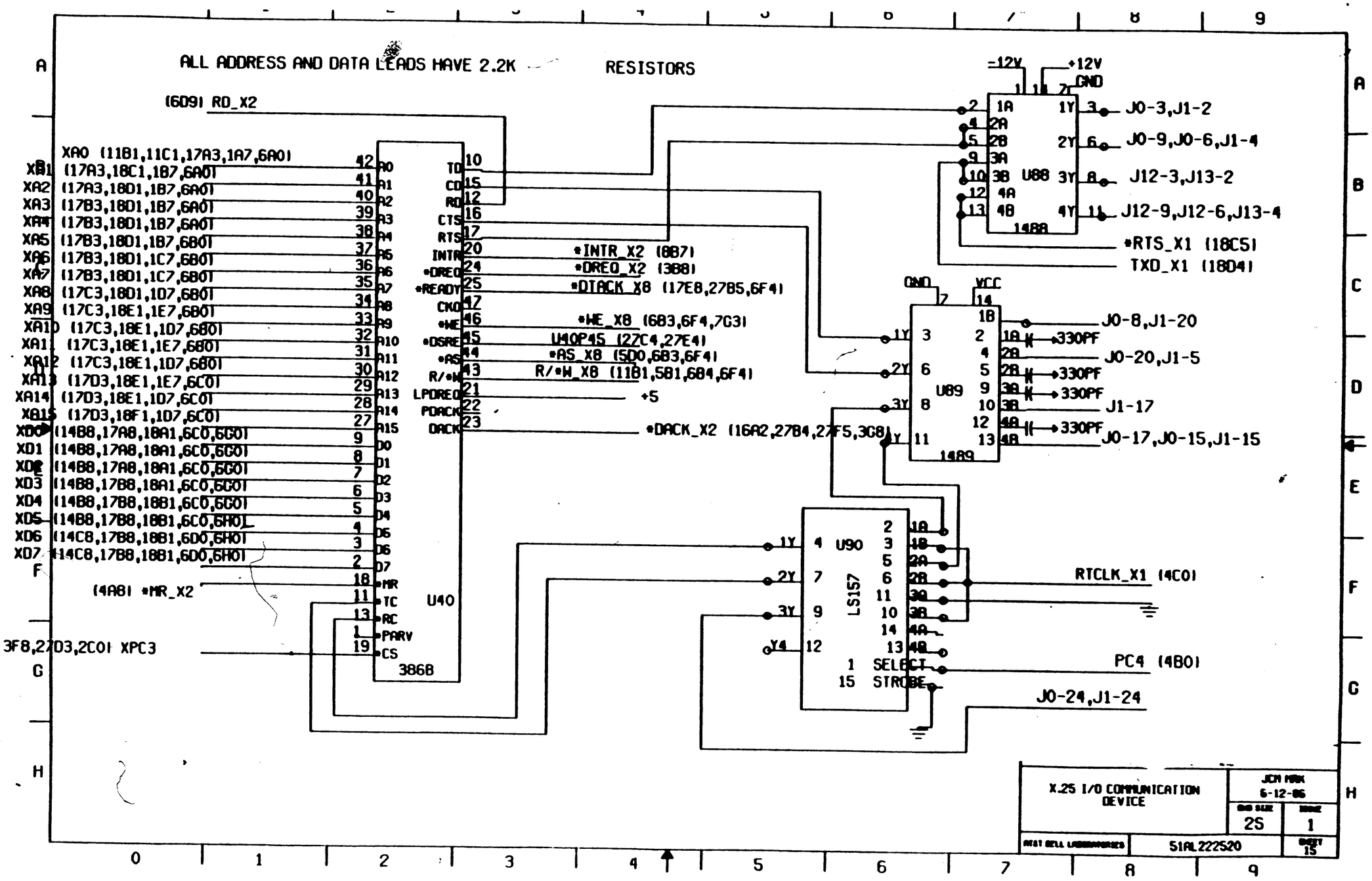
U33 LS245
(2E8) B3E
(8E3) BUFDIR
XD0 (15E0,17A8,18A1,6C0,6G0)
XD1 (15E0,17A8,18A1,6C0,6G0)
XD2 (15E0,17A8,18A1,6C0,6G0)
XD3 (15E0,17B8,18A1,6C0,6G0)
XD4 (15E0,17B8,18B1,6C0,6G0)
XD5 (15E0,17B8,18B1,6C0,6H0)
XD6 (15F0,17B8,18B1,6D0,6H0)
XD7 (15F0,17B8,18B1,6D0,6H0)

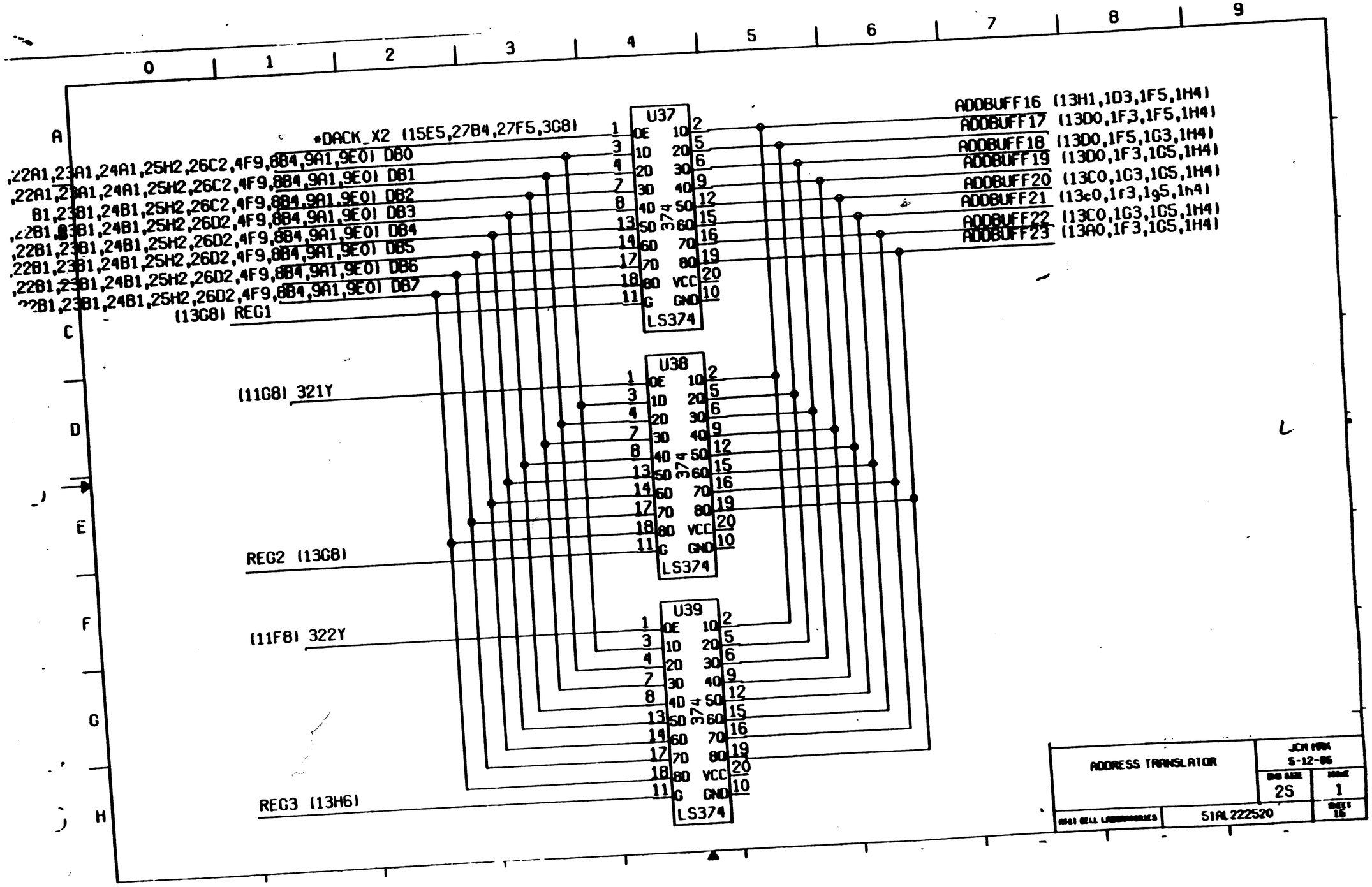
U34 LS245
(19C1,20C1,21C1,22C1,23C1,24C1,25H2) DB8
(19C1,20C1,21C1,22C1,23C1,24C1,25H2) DB9
(19C1,20C1,21C1,22C1,23C1,24C1,25H2) DB10
(19C1,20C1,21C1,22C1,23C1,24C1,25H2) DB11
(19C1,20C1,21C1,22C1,23C1,24C1,25H2) DB12
(19D1,20D1,21D1,22D1,23D1,24D1,25H2) DB13
(19D1,20D1,21D1,22D1,23D1,24D1,25H2) DB14
(19D1,20D1,21D1,22D1,23D1,24D1,25H2) DB15
(2D9) B4E
(8E3) BUFDIR

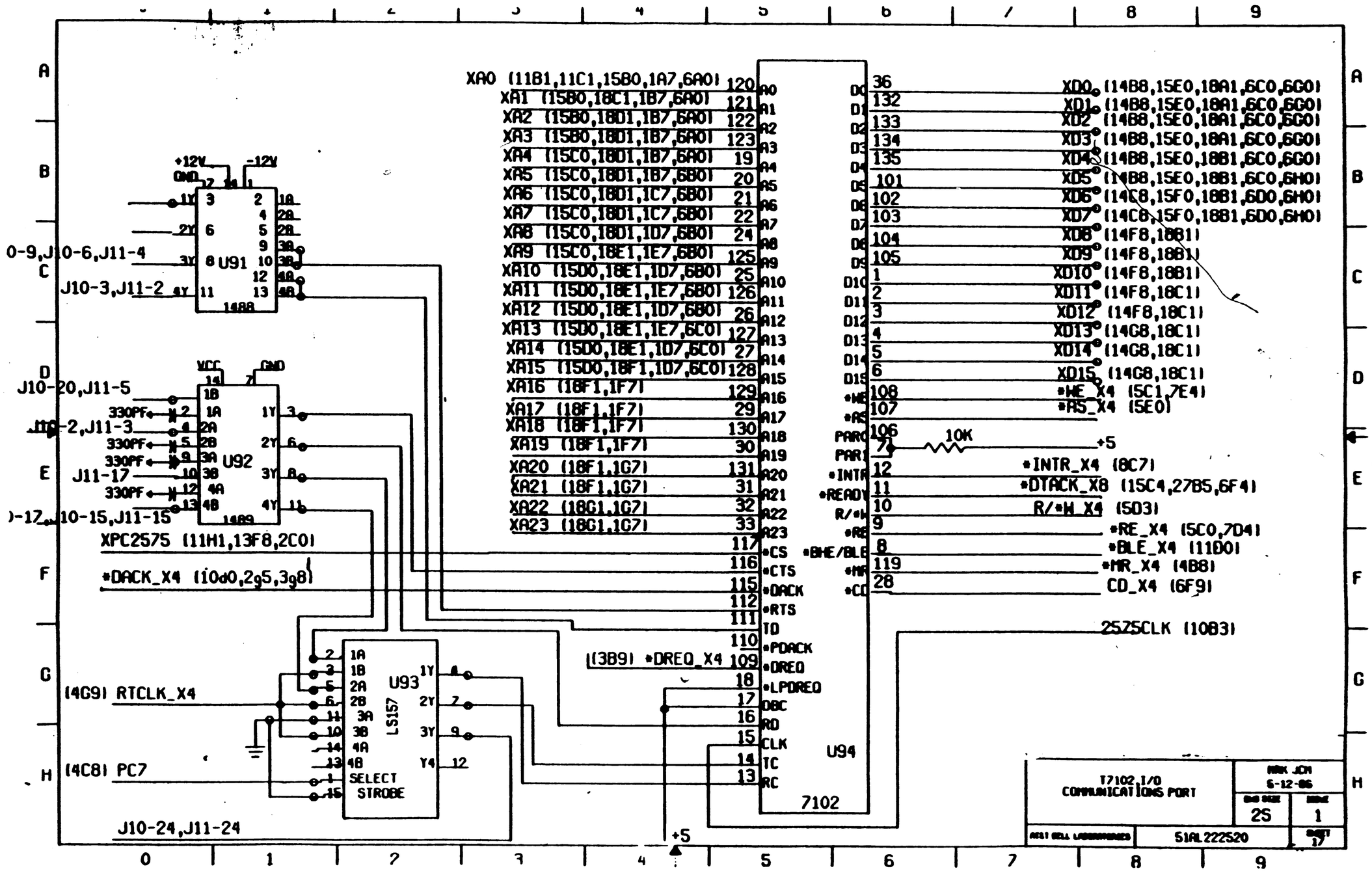
U5 LS245
MB8 (12B5,12D5) 2
MB9 (12B5,12D5) 3
MB10 (12B5,12D5) 4
MB11 (12B5,12D5) 5
MB12 (12B5,12D5) 6
MB13 (12B5,12D5) 7
MB14 (12B5,12E5) 8
MB15 (12B5,12E5) 9
BGACK (1C0,1F0,1H0,3H1) 19
R/*WBUS (25H2,26B6,5G7)
(19C1,20C1,21C1,22C1,23C1,24C1,25H2) DB8
(19C1,20C1,21C1,22C1,23C1,24C1,25H2) DB9
(19C1,20C1,21C1,22C1,23C1,24C1,25H2) DB10
(19C1,20C1,21C1,22C1,23C1,24C1,25H2) DB11
(19C1,20C1,21C1,22C1,23C1,24C1,25H2) DB12
(19D1,20D1,21D1,22D1,23D1,24D1,25H2) DB13
(19D1,20D1,21D1,22D1,23D1,24D1,25H2) DB14
(19D1,20D1,21D1,22D1,23D1,24D1,25H2) DB15

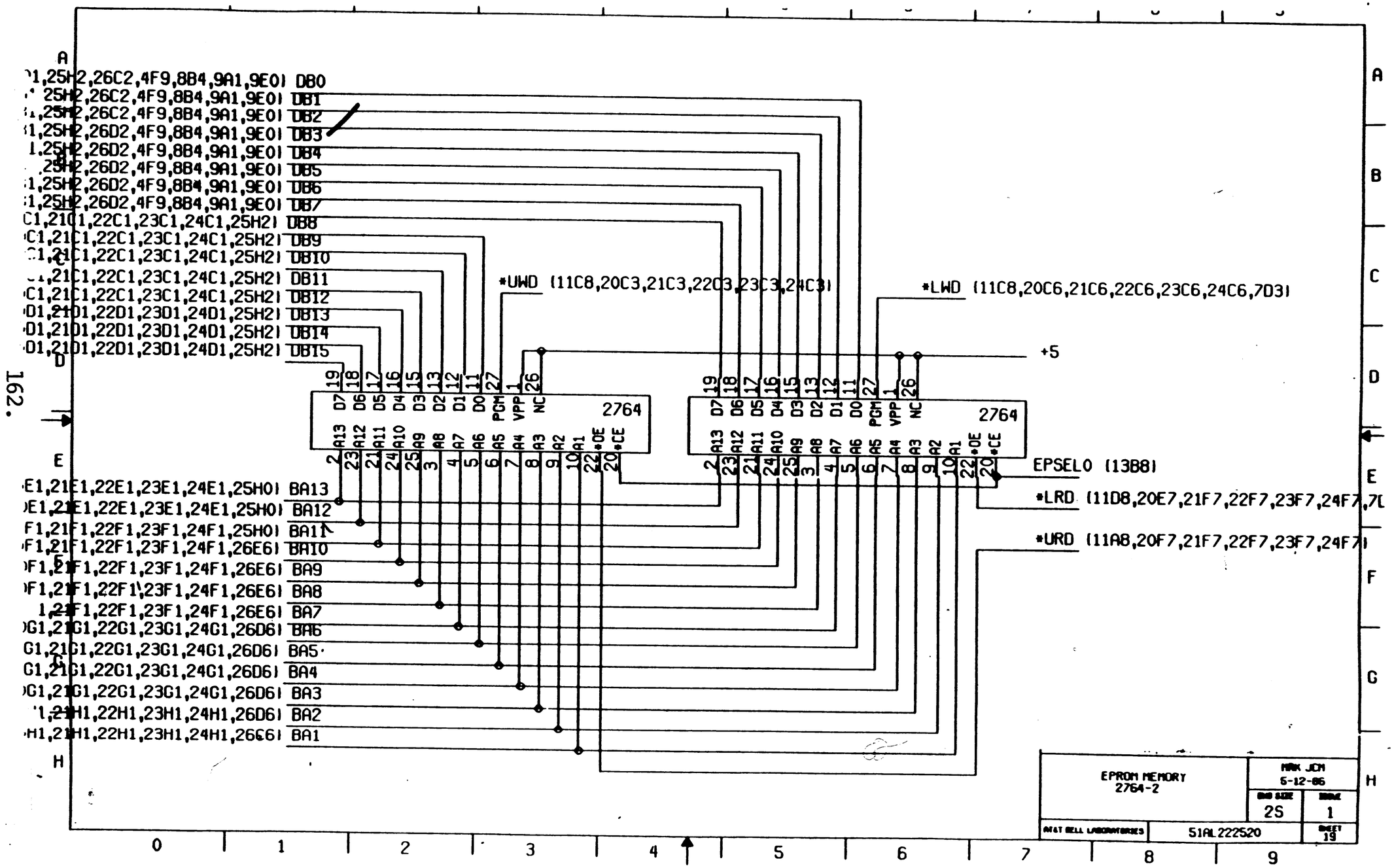
U35 LS245
B36E (10D5,1H6)
BUFDIR (8E3)
XD8 (17C8,18B1)
XD9 (17C8,18B1)
XD10 (17C8,18B1)
XD11 (17C8,18C1)
XD12 (17C8,18C1)
XD13 (17D8,18C1)
XD14 (17D8,18C1)
XD15 (17D8,18C1)

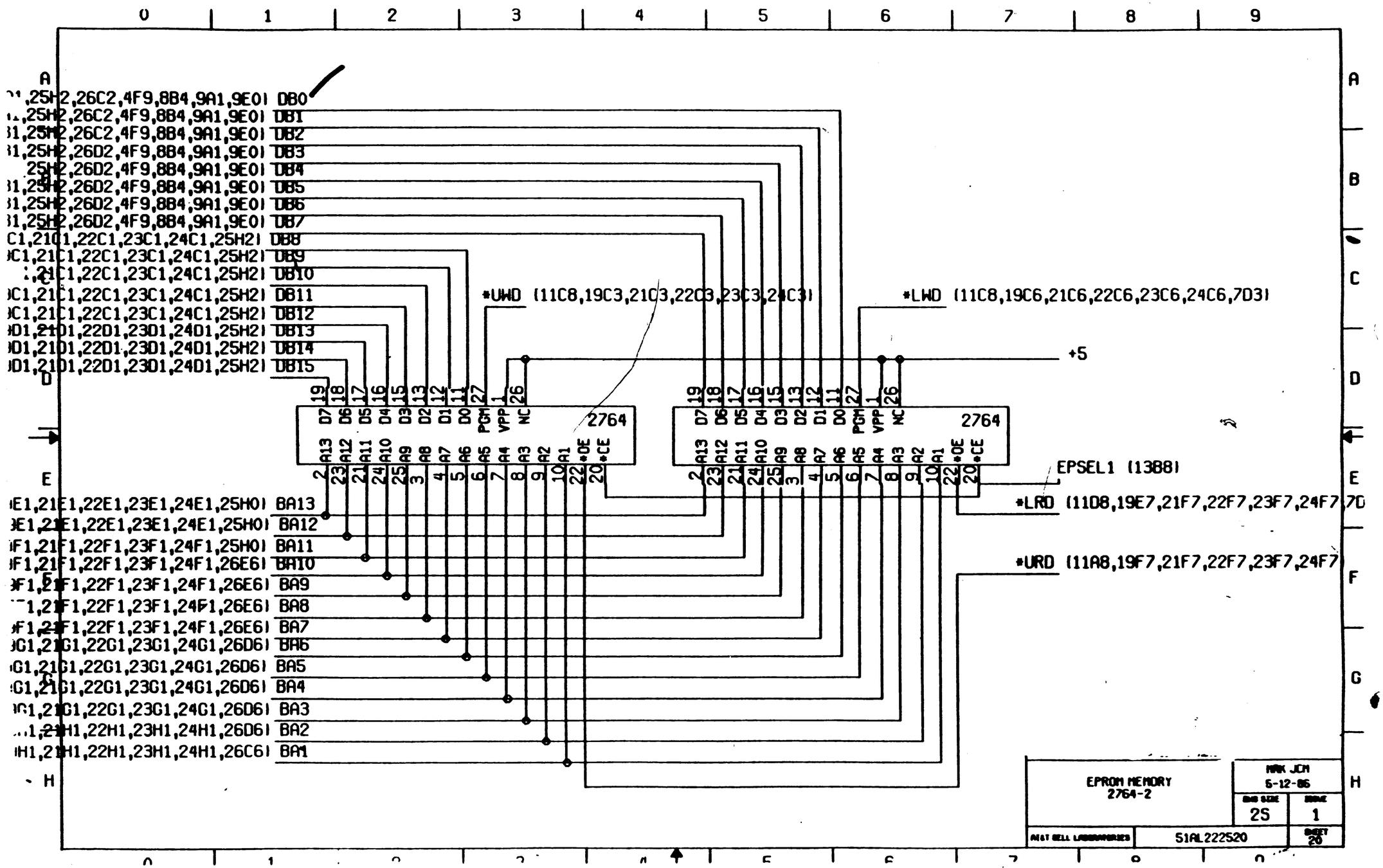
data buffers	JCM MRK 5-12-86	
	BUS SIZE 2S	SHEET 1
AT&T BELL LABORATORIES	51a1222520	SHEET 14

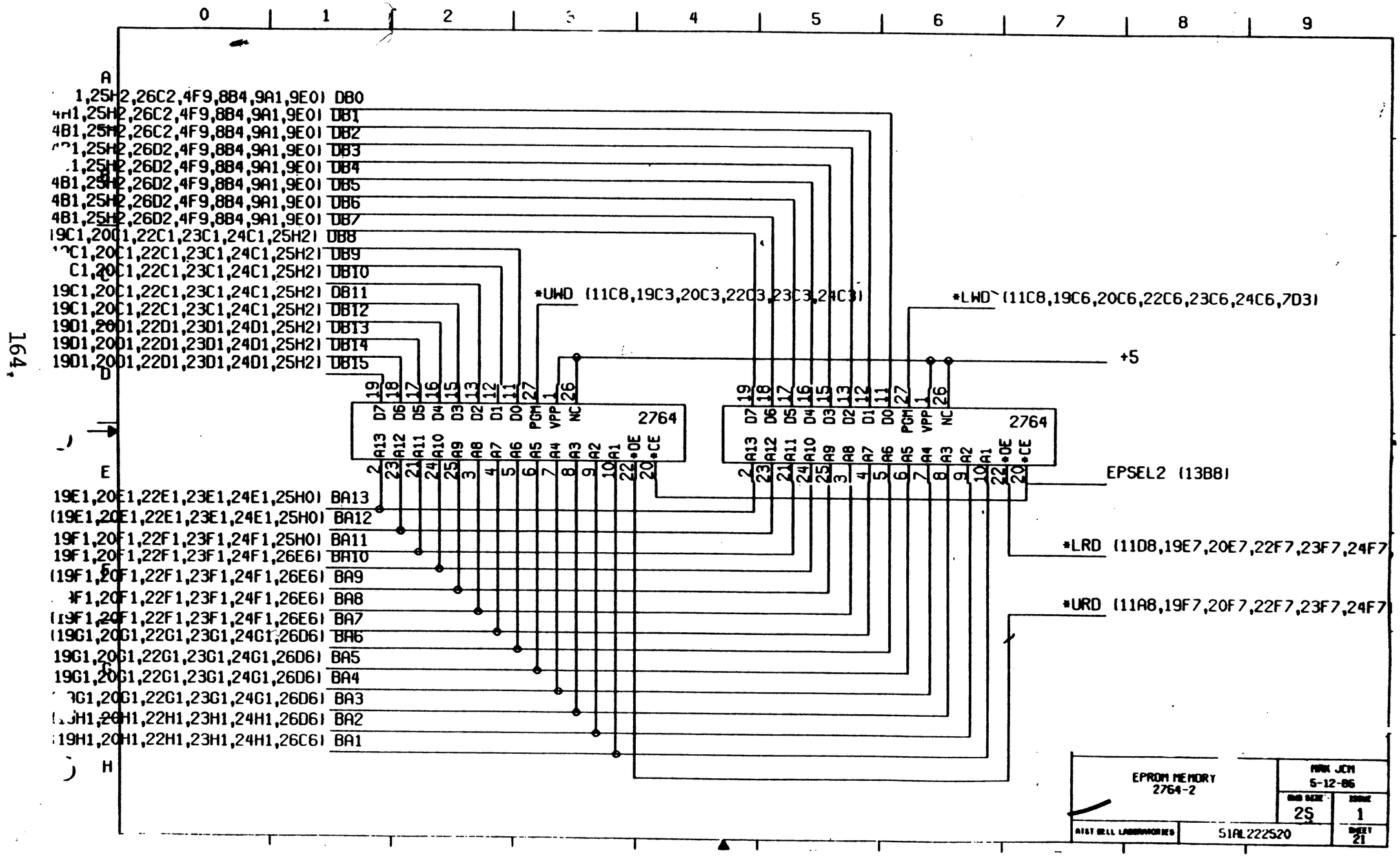

ALL ADDRESS AND DATA LEADS HAVE 2.2K RESISTORS
(6D9) RD_X2
XA0 (11B1,11C1,17A3,1A7,6A0)
XA1 (17A3,18C1,1B7,6A0)
XA2 (17A3,18D1,1B7,6A0)
XA3 (17B3,18D1,1B7,6A0)
XA4 (17B3,18D1,1B7,6A0)
XA5 (17B3,18D1,1B7,6B0)
XA6 (17B3,18D1,1C7,6B0)
XA7 (17B3,18D1,1C7,6B0)
XA8 (17C3,18D1,1D7,6B0)
XA9 (17C3,18E1,1E7,6B0)
XA10 (17C3,18E1,1D7,6B0)
XA11 (17C3,18E1,1E7,6B0)
XA12 (17C3,18E1,1D7,6B0)
XA13 (17D3,18E1,1E7,6C0)
XA14 (17D3,18E1,1D7,6C0)
XA15 (17D3,18F1,1D7,6C0)
XD0 (14B8,17A8,18A1,6C0,6G0)
XD1 (14B8,17A8,18A1,6C0,6G0)
XD2 (14B8,17A8,18A1,6C0,6G0)
XD3 (14B8,17B8,18A1,6C0,6G0)
XD4 (14B8,17B8,18B1,6C0,6G0)
XD5 (14B8,17B8,18B1,6C0,6H0)
XD6 (14C8,17B8,18B1,6D0,6H0)
XD7 (14C8,17B8,18B1,6D0,6H0)
(4A8) *MR_X2
3F8,27D3,2C0) XPC3
U40
386B
TD
CD
RD
CTS
RTS
INTR
*DREQ
*READY
CKO
*WE
*DSRE
*AS
R/*W
LPDREQ
PDACK
DACK
*MR
TC
RC
PARV
CS
*INTR_X2 (8B7)
*DREQ_X2 (3B8)
*DTACK_X8 (17E8,27B5,6F4)
*WE_X8 (6B3,6F4,7G3)
U40P45 (27C4,27E4)
*AS_X8 (5D0,6B3,6F4)
R/*W_X8 (11B1,5B1,6B4,6F4)
+5
*DACK_X2 (16A2,27B4,27F5,3G8)
=12V
+12V
GND
U88
1488
*RTS_X1 (18C5)
TXD_X1 (18D4)
J0-3,J1-2
J0-9,J0-6,J1-4
J12-3,J13-2
J12-9,J12-6,J13-4
U89
1489
VCC
330PF
J0-8,J1-20
J0-20,J1-5
J1-17
J0-17,J0-15,J1-15
U90
LS157
RTCLK_X1 (4C0)
PC4 (4B0)
J0-24,J1-24
X.25 I/O COMMUNICATION DEVICE
JCN MRK
6-12-86
SIAL222520
SHEET 15

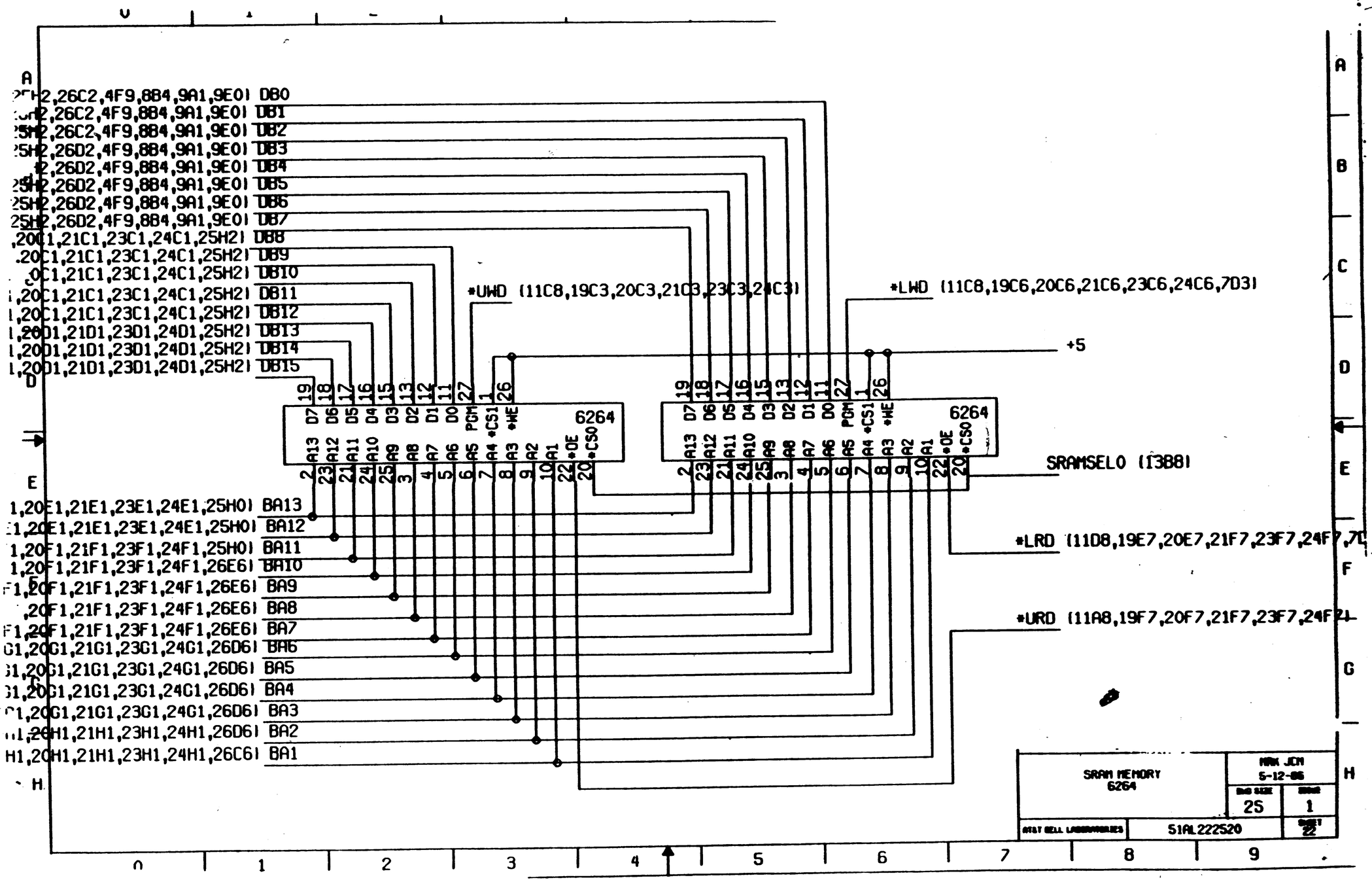

*DACK_X2 (15E5,27B4,27F5,3C8)
,22A1,23A1,24A1,25H2,26C2,4F9,8B4,9A1,9E0) DB0
,22A1,23A1,24A1,25H2,26C2,4F9,8B4,9A1,9E0) DB1
B1,23B1,24B1,25H2,26C2,4F9,8B4,9A1,9E0) DB2
,22B1,23B1,24B1,25H2,26D2,4F9,8B4,9A1,9E0) DB3
,22B1,23B1,24B1,25H2,26D2,4F9,8B4,9A1,9E0) DB4
,22B1,23B1,24B1,25H2,26D2,4F9,8B4,9A1,9E0) DB5
,22B1,23B1,24B1,25H2,26D2,4F9,8B4,9A1,9E0) DB6
,22B1,23B1,24B1,25H2,26D2,4F9,8B4,9A1,9E0) DB7
(13C8) REG1
(11G8) 321Y
REG2 (13C8)
(11F8) 322Y
REG3 (13H6)
U37
U38
U39
OE
D
G
VCC
GND
374
LS374
ADDBUFF16 (13H1,1D3,1F5,1H4)
ADDBUFF17 (13D0,1F3,1F5,1H4)
ADDBUFF18 (13D0,1F5,1G3,1H4)
ADDBUFF19 (13D0,1F3,1G5,1H4)
ADDBUFF20 (13C0,1G3,1G5,1H4)
ADDBUFF21 (13c0,1f3,1g5,1h4)
ADDBUFF22 (13C0,1G3,1G5,1H4)
ADDBUFF23 (13A0,1F3,1G5,1H4)
ADDRESS TRANSLATOR
JCH MRK
5-12-86
SIZE
2S
DRWG
1
51AL222520
SHEET
16

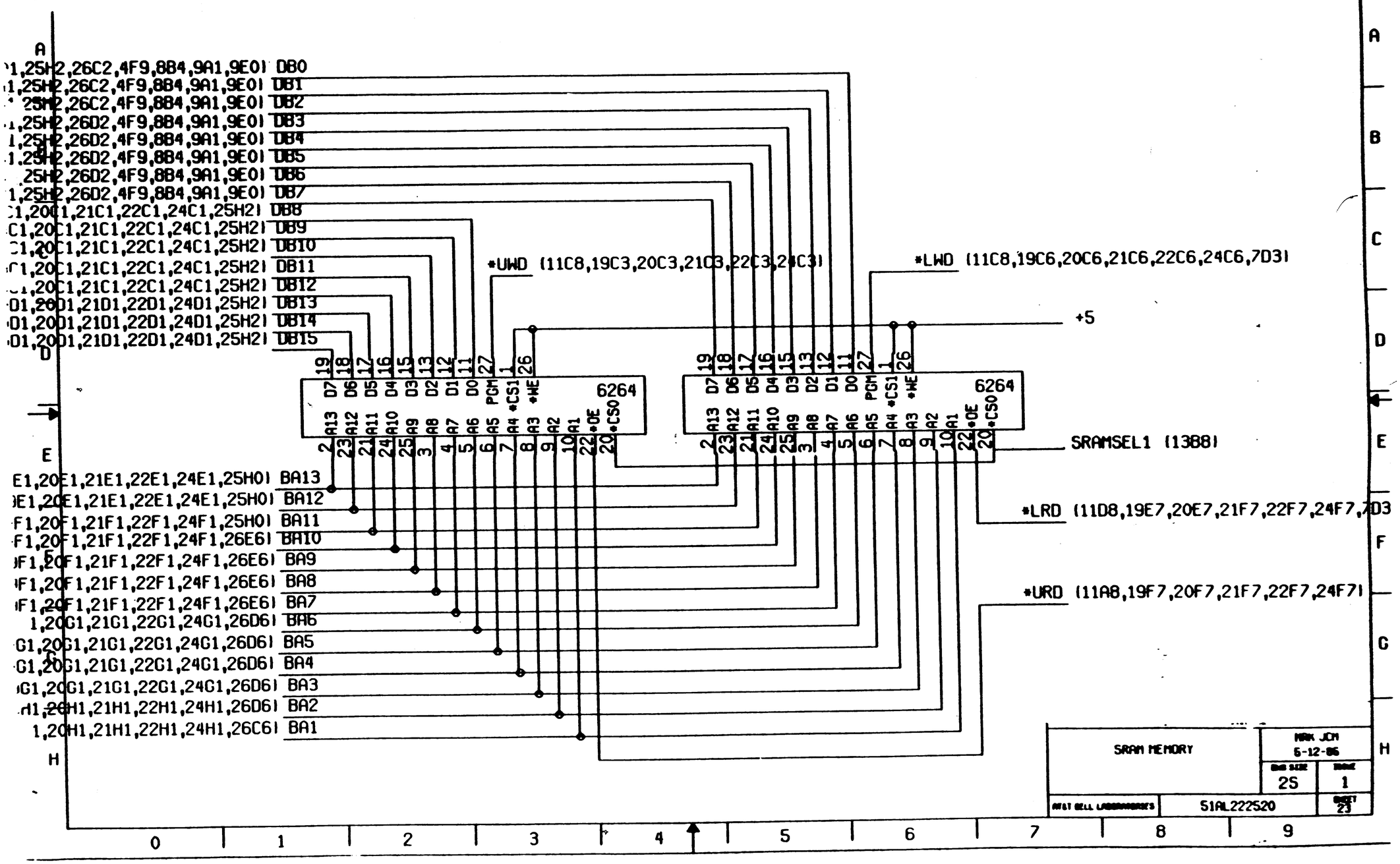

+12V
-12V
GND
U91
1488
0-9,J10-6,J11-4
J10-3,J11-2
U92
1489
VCC
GND
J10-20,J11-5
J10-2,J11-3
J11-17
J10-15,J11-15
330PF
XPC2575 (11H1,13F8,2C0)
*DACK_X4 (10d0,2g5,3g8)
U93
LS157
SELECT
STROBE
(4G9) RTCLK_X4
(4C8) PC7
J10-24,J11-24
XA0 (11B1,11C1,15B0,1A7,6A0)
XA1 (15B0,18C1,1B7,6A0)
XA2 (15B0,18D1,1B7,6A0)
XA3 (15B0,18D1,1B7,6A0)
XA4 (15C0,18D1,1B7,6A0)
XA5 (15C0,18D1,1B7,6B0)
XA6 (15C0,18D1,1C7,6B0)
XA7 (15C0,18D1,1C7,6B0)
XA8 (15C0,18D1,1D7,6B0)
XA9 (15C0,18E1,1E7,6B0)
XA10 (15D0,18E1,1D7,6B0)
XA11 (15D0,18E1,1E7,6B0)
XA12 (15D0,18E1,1D7,6B0)
XA13 (15D0,18E1,1E7,6C0)
XA14 (15D0,18E1,1D7,6C0)
XA15 (15D0,18F1,1D7,6C0)
XA16 (18F1,1F7)
XA17 (18F1,1F7)
XA18 (18F1,1F7)
XA19 (18F1,1F7)
XA20 (18F1,1G7)
XA21 (18F1,1G7)
XA22 (18G1,1G7)
XA23 (18G1,1G7)
*CS
*CTS
*DACK
*RTS
TD
*PDACK
*DREQ
*LPDREQ
DBC
RD
CLK
TC
RC
(3B9) *DREQ_X4
U94
7102
XD0 (14B8,15E0,18A1,6C0,6G0)
XD1 (14B8,15E0,18A1,6C0,6G0)
XD2 (14B8,15E0,18A1,6C0,6G0)
XD3 (14B8,15E0,18A1,6C0,6G0)
XD4 (14B8,15E0,18B1,6C0,6G0)
XD5 (14B8,15E0,18B1,6C0,6H0)
XD6 (14C8,15F0,18B1,6D0,6H0)
XD7 (14C8,15F0,18B1,6D0,6H0)
XD8 (14F8,18B1)
XD9 (14F8,18B1)
XD10 (14F8,18B1)
XD11 (14F8,18C1)
XD12 (14F8,18C1)
XD13 (14G8,18C1)
XD14 (14G8,18C1)
XD15 (14G8,18C1)
*WE_X4 (5C1,7E4)
*AS_X4 (5E0)
10K
+5
*INTR_X4 (8C7)
*DTACK_X8 (15C4,27B5,6F4)
R/*W_X4 (5D3)
*RE_X4 (5C0,7D4)
*BLE_X4 (11D0)
*MR_X4 (4B8)
CD_X4 (6F9)
2S2SCLK (10B3)
T7102 I/O COMMUNICATIONS PORT
MRK JCH 5-12-86
2S
1
51AL222520

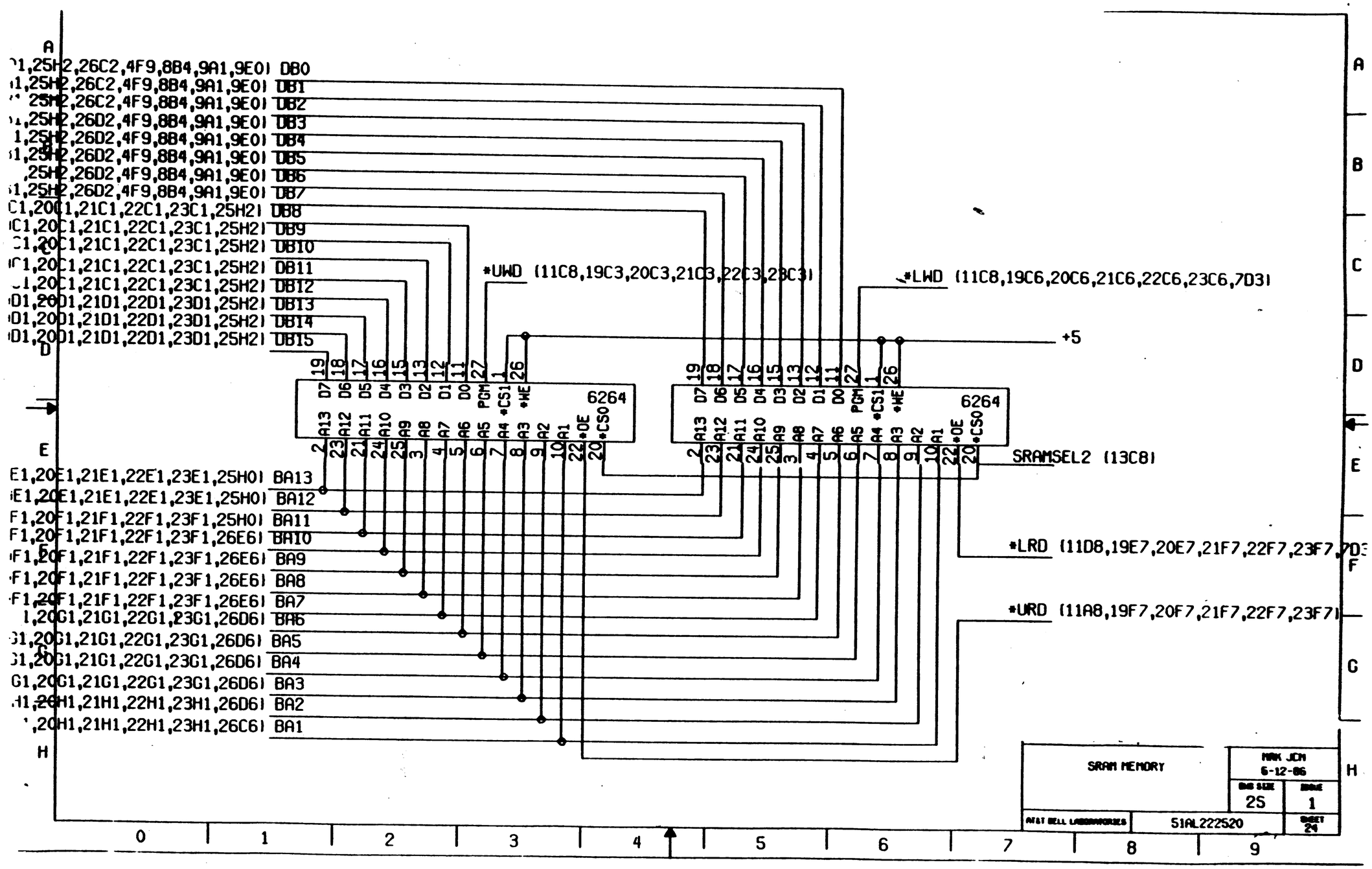

DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
DB8
DB9
DB10
DB11
DB12
DB13
DB14
DB15
*UWD (11C8,20C3,21C3,22C3,23C3,24C3)
*LWD (11C8,20C6,21C6,22C6,23C6,24C6,7D3)
+5
2764
BA13
BA12
BA11
BA10
BA9
BA8
BA7
BA6
BA5
BA4
BA3
BA2
BA1
EPSEL0 (13B8)
*LRD (11D8,20E7,21F7,22F7,23F7,24F7,7L
*URD (11A8,20F7,21F7,22F7,23F7,24F7
EPROM MEMORY
2764-2
MARK JOH
5-12-86
SIZE 2S
DWG 1
AT&T BELL LABORATORIES
51AL222520
SHEET 19


0 1 2 3 4 5 6 7 8 9
A B C D E F G H
,25H2,26C2,4F9,8B4,9A1,9E0) DB0
,25H2,26C2,4F9,8B4,9A1,9E0) DB1
1,25H2,26C2,4F9,8B4,9A1,9E0) DB2
1,25H2,26D2,4F9,8B4,9A1,9E0) DB3
25H2,26D2,4F9,8B4,9A1,9E0) DB4
1,25H2,26D2,4F9,8B4,9A1,9E0) DB5
1,25H2,26D2,4F9,8B4,9A1,9E0) DB6
1,25H2,26D2,4F9,8B4,9A1,9E0) DB7
C1,21C1,22C1,23C1,24C1,25H2) DB8
C1,21C1,22C1,23C1,24C1,25H2) DB9
21C1,22C1,23C1,24C1,25H2) DB10
C1,21C1,22C1,23C1,24C1,25H2) DB11
C1,21C1,22C1,23C1,24C1,25H2) DB12
D1,21D1,22D1,23D1,24D1,25H2) DB13
D1,21D1,22D1,23D1,24D1,25H2) DB14
D1,21D1,22D1,23D1,24D1,25H2) DB15
*UWD (11C8,19C3,21C3,22C3,23C3,24C3)
*LWD (11C8,19C6,21C6,22C6,23C6,24C6,7D3)
+5
2764
2764
D7 D6 D5 D4 D3 D2 D1 D0 PGM VPP NC
A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 *OE *CE
EPSEL1 (13B8)
*LRD (11D8,19E7,21F7,22F7,23F7,24F7,7D
*URD (11A8,19F7,21F7,22F7,23F7,24F7
E1,21E1,22E1,23E1,24E1,25H0) BA13
E1,21E1,22E1,23E1,24E1,25H0) BA12
F1,21F1,22F1,23F1,24F1,25H0) BA11
F1,21F1,22F1,23F1,24F1,26E6) BA10
F1,21F1,22F1,23F1,24F1,26E6) BA9
1,21F1,22F1,23F1,24F1,26E6) BA8
F1,21F1,22F1,23F1,24F1,26E6) BA7
G1,21G1,22G1,23G1,24G1,26D6) BA6
G1,21G1,22G1,23G1,24G1,26D6) BA5
G1,21G1,22G1,23G1,24G1,26D6) BA4
G1,21G1,22G1,23G1,24G1,26D6) BA3
1,21H1,22H1,23H1,24H1,26D6) BA2
H1,21H1,22H1,23H1,24H1,26C6) BA1
EPROM MEMORY
2764-2
MARK JCM
5-12-86
SIZE
2S
IMAGE
1
BELL LABORATORIES
51AL222520
SHEET
20


0 1 2 3 4 5 6 7 8 9
A
B
C
D
E
F
G
H
164
1,25H2,26C2,4F9,8B4,9A1,9E0) DB0
4B1,25H2,26C2,4F9,8B4,9A1,9E0) DB1
4B1,25H2,26C2,4F9,8B4,9A1,9E0) DB2
1,25H2,26D2,4F9,8B4,9A1,9E0) DB3
1,25H2,26D2,4F9,8B4,9A1,9E0) DB4
4B1,25H2,26D2,4F9,8B4,9A1,9E0) DB5
4B1,25H2,26D2,4F9,8B4,9A1,9E0) DB6
4B1,25H2,26D2,4F9,8B4,9A1,9E0) DB7
19C1,20C1,22C1,23C1,24C1,25H2) DB8
19C1,20C1,22C1,23C1,24C1,25H2) DB9
C1,20C1,22C1,23C1,24C1,25H2) DB10
19C1,20C1,22C1,23C1,24C1,25H2) DB11
19C1,20C1,22C1,23C1,24C1,25H2) DB12
19D1,20D1,22D1,23D1,24D1,25H2) DB13
19D1,20D1,22D1,23D1,24D1,25H2) DB14
19D1,20D1,22D1,23D1,24D1,25H2) DB15
*UWD (11C8,19C3,20C3,22C3,23C3,24C3)
*LWD (11C8,19C6,20C6,22C6,23C6,24C6,7D3)
+5
EPSEL2 (13B8)
*LRD (11D8,19E7,20E7,22F7,23F7,24F7
*URD (11A8,19F7,20F7,22F7,23F7,24F7
19E1,20E1,22E1,23E1,24E1,25H0) BA13
(19E1,20E1,22E1,23E1,24E1,25H0) BA12
19F1,20F1,22F1,23F1,24F1,25H0) BA11
19F1,20F1,22F1,23F1,24F1,26E6) BA10
(19F1,20F1,22F1,23F1,24F1,26E6) BA9
F1,20F1,22F1,23F1,24F1,26E6) BA8
(19F1,20F1,22F1,23F1,24F1,26E6) BA7
(19G1,20G1,22G1,23G1,24G1,26D6) BA6
19G1,20G1,22G1,23G1,24G1,26D6) BA5
19G1,20G1,22G1,23G1,24G1,26D6) BA4
G1,20G1,22G1,23G1,24G1,26D6) BA3
H1,20H1,22H1,23H1,24H1,26D6) BA2
19H1,20H1,22H1,23H1,24H1,26C6) BA1
D7 D6 D5 D4 D3 D2 D1 D0 PGM VPP NC
19 18 17 16 15 13 12 11 27 1 26
A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 •OE •CE
2 23 21 24 25 3 4 5 6 7 8 9 10 22 20
2764
EPROM MEMORY
2764-2
MRK JCH
5-12-86
SIZE 2S
ISSUE 1
AT&T BELL LABORATORIES
51AL222520
SHEET 21


*UWD (11C8,19C3,20C3,21C3,23C3,24C3)
*LWD (11C8,19C6,20C6,21C6,23C6,24C6,7D3)
+5
SRAMSEL0 (13B8)
*LRD (11D8,19E7,20E7,21F7,23F7,24F7,7
*URD (11A8,19F7,20F7,21F7,23F7,24F
6264
SRAM MEMORY
6264
MRK JCN
5-12-86
2S
1
51AL222520
22


*UWD (11C8,19C3,20C3,21C3,22C3,24C3)
*LWD (11C8,19C6,20C6,21C6,22C6,24C6,7D3)
+5
SRAMSEL1 (13B8)
*LRD (11D8,19E7,20E7,21F7,22F7,24F7,7D3
*URD (11A8,19F7,20F7,21F7,22F7,24F7)
6264
D7 D6 D5 D4 D3 D2 D1 D0 PGM *CS1 *WE
A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 *OE *CS0
DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7 DB8 DB9 DB10 DB11 DB12 DB13 DB14 DB15
BA13 BA12 BA11 BA10 BA9 BA8 BA7 BA6 BA5 BA4 BA3 BA2 BA1
SRAM MEMORY
MAK JCH
6-12-86
SIZE 2S
DWG NO 1
SHEET 23
SIAL222520


A
B
C
D
E
F
G
H
1,25H2,26C2,4F9,8B4,9A1,9E0) DB0
1,25H2,26C2,4F9,8B4,9A1,9E0) DB1
25H2,26C2,4F9,8B4,9A1,9E0) DB2
,25H2,26D2,4F9,8B4,9A1,9E0) DB3
1,25H2,26D2,4F9,8B4,9A1,9E0) DB4
1,25H2,26D2,4F9,8B4,9A1,9E0) DB5
,25H2,26D2,4F9,8B4,9A1,9E0) DB6
1,25H2,26D2,4F9,8B4,9A1,9E0) DB7
C1,20C1,21C1,22C1,23C1,25H2) DB8
C1,20C1,21C1,22C1,23C1,25H2) DB9
C1,20C1,21C1,22C1,23C1,25H2) DB10
C1,20C1,21C1,22C1,23C1,25H2) DB11
C1,20C1,21C1,22C1,23C1,25H2) DB12
D1,20D1,21D1,22D1,23D1,25H2) DB13
D1,20D1,21D1,22D1,23D1,25H2) DB14
D1,20D1,21D1,22D1,23D1,25H2) DB15
*UWD (11C8,19C3,20C3,21C3,22C3,23C3)
*LWD (11C8,19C6,20C6,21C6,22C6,23C6,7D3)
+5
6264
D7 D6 D5 D4 D3 D2 D1 D0 PGM *CS1 *WE
A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 *OE *CS0
SRAMSEL2 (13C8)
*LRD (11D8,19E7,20E7,21F7,22F7,23F7,7D3
*URD (11A8,19F7,20F7,21F7,22F7,23F7)
E1,20E1,21E1,22E1,23E1,25H0) BA13
E1,20E1,21E1,22E1,23E1,25H0) BA12
F1,20F1,21F1,22F1,23F1,25H0) BA11
F1,20F1,21F1,22F1,23F1,26E6) BA10
F1,20F1,21F1,22F1,23F1,26E6) BA9
F1,20F1,21F1,22F1,23F1,26E6) BA8
F1,20F1,21F1,22F1,23F1,26E6) BA7
1,20G1,21G1,22G1,23G1,26D6) BA6
G1,20G1,21G1,22G1,23G1,26D6) BA5
G1,20G1,21G1,22G1,23G1,26D6) BA4
G1,20G1,21G1,22G1,23G1,26D6) BA3
H1,20H1,21H1,22H1,23H1,26D6) BA2
,20H1,21H1,22H1,23H1,26C6) BA1
SRAM MEMORY
MRK JCM
6-12-86
SIZE
2S
DRAWING
1
AT&T BELL LABORATORIES
51AL222520
SHEET
24
0 1 2 3 4 5 6 7 8 9